Intelligent Automation and Soft Computing, Vol. 11, No. 4, pp. 217-234, 2005
Copyright © 2005, TSI® Press
Printed in the USA. All rights reserved

A STUDY OF FIVE PARALLEL APPROACHES TO A GENETIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

L. WANG[1], A. A. MACIEJEWSKI[2], H. J. SIEGEL[2,3], V. P. ROYCHOWDHURY[4], AND B. D. ELDRIDGE[2]

[1]Microsoft Corporation
Redmond, WA 98052-6399

[2]Electrical and Computer Engineering Department
[3]Computer Science Department
Colorado State University
Fort Collins, CO 80526-1373
email:aam@colostate.edu
(CORRESPONDING AUTHOR)

[4]Electrical Engineering Department
UCLA
Los Angeles, CA 90095-1594

ABSTRACT—This paper presents a comparative study of five different coarse-grained parallel genetic algorithms (PGAs) using the traveling salesman problem as the case application. All of these PGAs are based on the same baseline serial genetic algorithm, implemented on the same parallel machine (IBM SP2), tested on the same problem instances, and started from the same set of initial populations. Based on these experiments, a PGA that combines a new subtour technique with a known migration approach is identified to be the best for the traveling salesman problem among the five PGAs being compared.

Key Words: Parallel Genetic Algorithms; Traveling Salesman Problem; Memetic Algorithms

1. INTRODUCTION

Serial genetic algorithms (SGAs) are a promising search heuristic for finding near-optimal solutions in large search spaces (e.g., [1]). To reduce the large amount of computation time associated with SGAs, parallel GAs (PGAs) have been proposed. PGAs have also been used to solve larger problems and to find better solutions.

There exists a large body of literature that discusses the parallelization of genetic algorithms for many different applications, e.g., [2-16]. Previous comparative studies have contrasted the performance of PGAs with and without migration, demonstrating that migration, in general, results in superior performance, e.g., [11, 13]. One of the features that distinguishes this paper from previous work is that it compares four conceptually different coarse-grain parallelization techniques, including two new approaches along with the standard independent and migration PGAs. In addition, it compares a PGA that is a hybrid of a new and a standard approach. All of these comparisons are done using a common framework.

The traveling salesman problem (TSP), described in Section 2, is used as the basis of this case study. To make fair comparisons, all of these PGAs are tested on the same set of TSP instances (listed in Section 2), implemented on the same parallel machine (the IBM SP2 described in Section 3), based on the same baseline SGA (presented in Section 4), and started from the same set of initial populations.

The PGAs studied are two existing approaches (the independent PGA and the migration PGA), two new approaches (the partitioned PGA and the segmentation PGA), and a hybrid approach (the segmentation-migration PGA). The independent PGA executes an independent SGA on each processor. The migration PGA extends the independent PGA by performing periodic migration of chromosomes among processors. The partitioned PGA partitions the problem space and has each processor search a disjoint partition. In the segmentation PGA, processors iteratively operate on TSP subtours (chromosome segments) and then recombine subtours into larger subtours until full tours (full chromosomes) are formed, at which point the independent approach is followed. The segmentation-migration PGA combines the segmentation and migration approaches. These PGA approaches are described in more detail in Section 5.

Extensive experiments were conducted to quantify each PGA's ability to find quality solutions and to test how quickly the PGAs can find solutions of similar quality. Experimental results showed that the four non-partitioned PGAs studied found high quality solutions (within 1% of the best known solutions). Inter-processor communications that migrate the locally best chromosomes among the processors shortened the total execution time. Using migration together with chromosome segmentation and recombination (the segmentation-migration approach) further reduced this time. These experiments are presented and analyzed in Section 6.

2. THE TRAVELLING SALESMAN PROBLEM

The traveling salesman problem (<u>TSP</u>) is a well-known NP-hard combinatorial optimization problem [17]. It can be stated as follows. There are $\underline{C}$ cities, which are numbered from 0 to $C - 1$. The distance from city i to city j is known to be $\underline{d_{ij}}$, where $0 \le i, j < C$ and $d_{ij} = 0$ if $i = j$. A <u>tour</u> is a path that starts from a city, visits each city exactly once, and goes back to the starting city. Mathematically, a tour can be expressed by a vector $\underline{t}$ of C elements. Each of the elements in t represents a city, i.e., $0 \le t(i) < C$ and $t(i) \ne t(j)$ if $i \ne j$. The tour starts from $t(0)$, visits cities in the order they appear in t, and then goes back to $t(0)$ after visiting $t(C-1)$. The goal of the TSP is to find a tour t with the minimum <u>tour length</u>, i.e., to minimize

$$\sum_{i=0}^{C-2} d_{t(i)t(i+1)} + d_{t(C-1)t(0)} \tag{1}$$

Because the tours are closed (circular), the same tour can be represented by multiple t-vectors that are rotations of one another. Thus, the vectors t_1 and t_2 represent the same tour if $t_1(i) = t_2((i + j) \text{ modulo } C)$ for any fixed integer j, for all i, $0 \le i < C$. If the distance between any two cities is independent of the travel direction, i.e., $d_{ij} = d_{ji}$ for all i and j, the TSP is a <u>symmetric TSP</u>. Otherwise, the TSP is <u>asymmetric</u>. For symmetric TSPs, the tour t_1 must be the same length as t_2 if t_1 is just a reversal of t_2, i.e., $t_1(i) = t_2(C-1-i)$, for all i, $0 \le i < C$. Furthermore, these rotation and symmetric properties can be combined to form equivalent tours.

A subset of the symmetric TSP is called the <u>Euclidean TSP</u>, where each city is represented by a set of coordinates. The distance between any two cities is simply the Euclidean distance. This research uses the same five instances of the 100-city Euclidean TSP to evaluate the performance of all the PGAs. In particular, the five 100-city Euclidean TSP instances with shortest known tour lengths from [18] are used as test cases. They are Krolak A (<u>KA</u>), Krolak C (<u>KC</u>), Krolak D (<u>KD</u>), <u>Reinelt</u> (or <u>RE</u> in tables), and the <u>lattice</u> (or <u>LT</u> in tables) TSPs. Among them only the lattice TSP is not randomly generated. In the lattice TSP, the cities are configured as a 10 x 10 lattice with unit distance between any pair of adjacent cities on the vertical and horizontal directions. The shortest possible tour for the lattice TSP has a length of 100.

3. THE IBM SP2 PARALLEL PROCESSING SYSTEM

An IBM RISC System/6000 POWERparallel 2 System (<u>SP2</u>) at the Purdue University Computing Center is used as the hardware platform to evaluate the PGAs being studied [19, 20]. The SP2 parallel processing system is a distributed-memory MIMD machine. The Purdue IBM SP2 consists of 18 processing elements (<u>PEs</u>) (i.e., processor-memory pairs) and an interconnection network. Among the PEs, 16 of them are ‘‘thin nodes,’’ with a 67-MHz CPU, 256 MBytes of main memory, two MBytes of second-level cache, and two GBytes of local disk space. All SGA experiments were conducted using one thin node; all PGA experiments were conducted using 16 thin nodes. To describe the PGAs in a general way, $\underline{N}$ is used as the number of PEs. The interconnection network of the IBM SP2 supports message passing among these PEs. The IBM SP2 provides the C Set++ compiler and the MPI message passing libraries [21, 22]. In this research, algorithms are coded using C and MPI routines.

4. SELECTING A SERIAL BASELINE GENETIC ALGORITHM FOR THE TSP

The purpose of this research is to comparatively study PGAs, rather than improving SGAs for a given application problem such as the TSP. Hence, the first step of this research is to choose an existing SGA from the literature to be the baseline SGA. For fair comparisons, all the PGAs studied are developed based on this same baseline SGA.

A variety of SGAs for the TSP have been proposed in the GA research literature (e.g., [4, 23-35]). From a search of this literature, it was determined that the SGA in [30] was among the best SGAs for the TSP (referred to as Jog's SGA in this work). A slightly modified version of Jog's SGA is used as the baseline SGA, which is outlined in Figure 1. The chromosome used in these SGAs, to represent a tour of the *C* cities, is the vector *t* defined in Section 2. At each iteration of the baseline SGA, three steps are executed in the following order: the modified Heuristic crossover step, the 2-opt step, and the modified Or-opt step. These modified steps incorporate local improvement techniques, classifying this approach as a "memetic" algorithm [36, 37]. These additions to the classic GA can improve both the speed at which the algorithm finds solutions and the quality of those solutions.

```
initial population generation; /* randomly generates
     a predefined number of unique chromosomes */
evaluation; /* obtain fitness value (tour length)
                 for each generated chromosome */
while(stopping criteria not met)
{  modified Heuristic crossover;
   2-opt;
   modified Or-opt;
   evaluation; /* only evaluates those that are
       either offspring of the crossover step or
       modified in the 2-opt or Or-opt steps */
}
output best chromosome;
```

Figure 1. Outline of the baseline SGA algorithm.

The Heuristic crossover step [30] randomly picks 50% of the chromosomes from the population. The picked chromosomes are then randomly paired and two offspring chromosomes are generated from each pair. The Heuristic crossover randomly picks a city as the starting point for each offspring tour. Then the Heuristic crossover operation extends the current partially constructed offspring tour by trying to add the shorter parental edge leading from the current last city on this partial offspring tour. Let the current last city on this partial offspring tour be *c*. Three different situations are considered in the following order when deciding which city to add to the current partial offspring tour:

(1) when the next city of *c* on both parent tours is not on the current partial offspring tour: the next city of *c* with the shorter distance from *c* is selected and added to the partial offspring tour;
(2) when the next city of *c* on exactly one parent tour is already on the current partial offspring tour: then the next city of *c* on the other parent tour is selected and added to the partial offspring tour; or
(3) when the next city of *c* on both parent tours is already on the current partial offspring tour: a random city is selected from those that are not yet on the partial offspring tour, and added to the partial offspring tour.

After this crossover operation, the two offspring tours replace the two parent tours.

The following changes to the Heuristic crossover step in Jog's SGA are made to add elitism to the baseline SGA. (It is important to have elitism in a GA [38], i.e., the currently best chromosome is always preserved and passed to the next generation.) Each pair of parent chromosomes generate only one offspring chromosome instead of two and this offspring replaces the parent tour with the longer length. By doing this, it guarantees that the currently best chromosome will be kept and passed to the next generation by this modified Heuristic crossover.

The 2−opt step [30] randomly selects one half of the remaining chromosomes of the population (25% of the total chromosomes). On each selected tour, ten 2-opt attempts are made to try to shorten the tour length, each time randomly picking two pairs of adjacent cities on the tour. The 2-opt operator takes these two pairs of adjacent cities (*a*, *b*) and (*c*, *d*) from a tour. If $d_{ab} + d_{cd} > d_{ac} + d_{bd}$, then the paths between cities *a*, *b* and cities *c*, *d* on the tour are removed and replaced by the new paths between cities *a*, *c* and cities *b*, *d*. The new tour is guaranteed to become shorter than the original tour. Figure 2 illustrates this 2-opt operation. The dotted lines between cities *b* to *c*, and cities *d* to *a* indicate that there may be other cities on the subtours between these cities. Note that the direction of traversing the subtour from b to c is reversed by the 2-opt operation (recall from Section 2 that symmetric TSPs are being considered here).

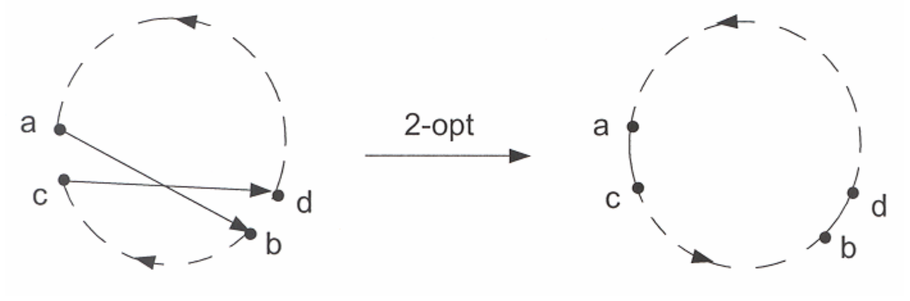


Figure 2. An example of 2-opt local improvements.

The Or−opt step [30] applies the Or-opt local improvement operator to the rest of the chromosomes of the population (25% of the total chromosomes). In each chromosome selected by this step, first all three-city subtours are chosen one by one. There are a total of *C* three-city subtours, the *i*-th ($0 \leq i < C$) of which contains cities $t(i)$, $t((i+1)$ modulo $C)$, and $t((i+2)$ modulo $C)$. If the subtour considered can be relocated between two other adjacent cities to form a shorter length tour, the appropriate tour changes are made.

Figure 3 illustrates an example of three-city Or-opt improvement. In this example, a three-city subtour (*d*, *e*, *f*) is being considered for relocation between a pair of adjacent cities *a* and *b*. If $d_{ab} + d_{cd} + d_{fg} > d_{af} + d_{db} + d_{cg}$, the three-city-subtour (*d*, *e*, *f*) will be inserted (in reverse order) between cities *a* and *b*, and cities *c* and *g* will be connected, resulting in a shorter length tour.

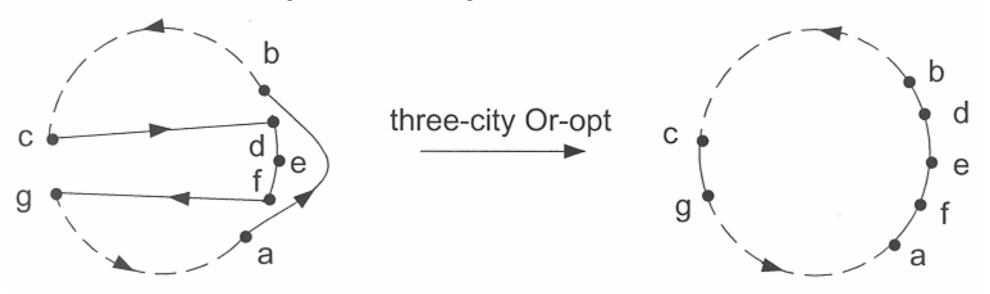


Figure 3. An example of three-city Or-opt local improvements.

In [30], for each *s*-city subtour (where *s* is first three, then two, and then one), one pair of adjacent cities (not on the subtour) is randomly chosen for consideration of the *s*-city subtour relocation. In the baseline SGA, for each *s*-city subtour (*s* being 3 first, then 2, and then 1) of a chosen tour, rather than only attempts to relocate this subtour between one pair of cities that are randomly chosen, the modified Or-opt step considers all possible pairs of cities for the relocation exhaustively. The modified Or-opt step calculates the resulting tour length for each possible relocation. The *s*-city subtour is put between a pair of cities (which could be its current position) that results in the shortest tour length.

In [30], the currently shortest tour is preserved in the 2-opt and Or-opt local improvements because these improvements never make the lengths of any tours longer. However, it is possible for the currently shortest tour to be replaced by a longer offspring tour in a Heuristic crossover operation (an example is given in [39]). A value-based roulette-wheel selection step is used in Jog's SGA to make sure that a shorter length tour has a better chance to be passed to the next generation. This selection step is not used in the baseline SGA algorithm because an implicit selection occurs at each modified Heuristic crossover operation, i.e., the shorter length parent tour is kept and the longer parent tour is replaced by the offspring tour.

Each run of Jog's SGA stops when the population converges, i.e., all chromosomes in the current population have the same fitness value (the same tour length). The baseline SGA runs stopped when the best solution was not improved for 150 iterations.

To test the baseline SGA, a total of 50 runs were conducted on each TSP instance, each starting with a different 128-tour initial population that was randomly generated. When the chromosomes were randomly generated, it was guaranteed that there were no identical tours in the initial population. In the initial populations, each tour started from the same city (city 0) to make it convenient to check whether any two tours were identical. As the evolution progressed, each chromosome could start from a different city and this starting city could be changed as a result of the genetic operations.

All the baseline SGA runs were executed on a single thin PE of the IBM SP2. Define the percentage quality difference, *D*, of one run of a given GA (SGA or PGA) compared to the best known solution of that TSP to be

$$D = \frac{\text{length of shortest tour found by GA} - \text{length of shortest known tour}}{\text{length of shortest known tour}} \times 100\% \qquad (2)$$

A smaller *D* indicates a better solution. If *D* = 0, the run found a tour with the shortest known length. Table I shows the performance of the baseline SGA in terms of solution quality (mean (μ) and maximum of *D*) and algorithm execution time (mean (μ) and median). It can be seen from the table that all tours the baseline SGA found were all less than 1% longer than the best known solutions. For the KA TSP and the KC TSP, all the runs found tours with the shortest known lengths. The solutions found for the KD TSP and the Reinelt TSP were also very close to the best known solutions, with maximum *D*s of 0.05% and 0.18%, respectively. On the lattice TSP, the baseline SGA was able to find tours that were either the shortest or the second shortest.

Table I. The performance of the baseline SGA using 128 chromosomes.

TSP	D (%)		execution time (sec)	
	μ	max	μ	median
KA	0.00	0.00	399	394
KC	0.00	0.00	419	419
KD	0.04	0.05	459	452
RE	0.09	0.18	490	466
LT	0.31	0.83	393	359

In [30], the KA TSP and lattice TSP instances were also used. Jog's SGA used 100 chromosomes, slightly smaller than the population size of 128 used in Table I. The ten-run average *D*s for the KA TSP and the lattice TSP in [30] were 1.37% and 0.40%, respectively, and the worst *D*s were 3.62% and 0.83%, respectively. Thus, the baseline SGA achieved better performance on these TSP instances. However, it typically took longer to execute than Jog's SGA because of the exhaustive nature of the modified Or-opt. For the PGA study, it was decided to use the baseline SGA as a basis because it could achieve results within 1% of the best known solutions for the TSPs considered.

5. A COMPARATIVE STUDY OF PGAs

5.1 Overview

In this research, a total of five different PGAs were developed based on the same baseline SGA. They are (1) the PGA using independent SGAs, (2) the PGA with chromosome migrations, (3) the PGA with search space partitioning, (4) the PGA with tour segmentation and recombination, and (5) the PGA that combines (2) and (4).

5.2 PGA using independent SGAs

The simplest way of parallelizing a GA is to execute multiple copies of the same SGA, one on each PE. Each of the 16 PEs starts with a different initial subpopulation (128/16 chromosomes), and evolves and stops independently. The complete PGA halts when all PEs stop. There are no inter-PE communications among any of these independent genetic evolutions at any time during the PGA execution.

This approach was named partitioned PGA in [13]. However, in order to not confuse it with the proposed PGA that partitions the search space, in this research this algorithm will be referred to as the independent PGA approach (because it consists of multiple independent executions of the baseline SGA). The advantage of the independent PGA approach is that each PE starts with an independent subpopulation. Such subpopulation diversity reduces the chance that all PEs prematurely converge to the same poor quality solution. This approach is equivalent to simply taking the best solution after multiple executions of the SGA on different initial populations.

5.3 PGA with chromosome migrations

The second PGA tested, the migration PGA approach, augments the independent approach with periodic chromosome migrations among the PEs. In this comparative study, the migration approach uses the chromosome migration scheme developed in [11]. Figure 4 shows the algorithm for the migration approach.

```
each PE executes the following:
       initial subpopulation generation;
       evaluation;
       while(stopping criteria not met){
              modified Heuristic crossover;
              2-opt;
              modified Or-opt;
              evaluation;
              perform chromosome migration
                     every fifth iteration;
       }
       output best chromosome;
```

Figure 4. Outline of the migration approach.

In the independent approach, it is possible for different PEs to prematurely converge to different suboptimal solutions, some of which may be of poor quality. Chromosome migration is a way of coordinating multiple evolutions (one on each PE) to prevent premature convergence at any local optima of poor quality by spreading globally high quality solutions among all PEs. Furthermore, the migration approach can speed up the slowly evolving subpopulations by introducing chromosomes that are better than the locally best ones.

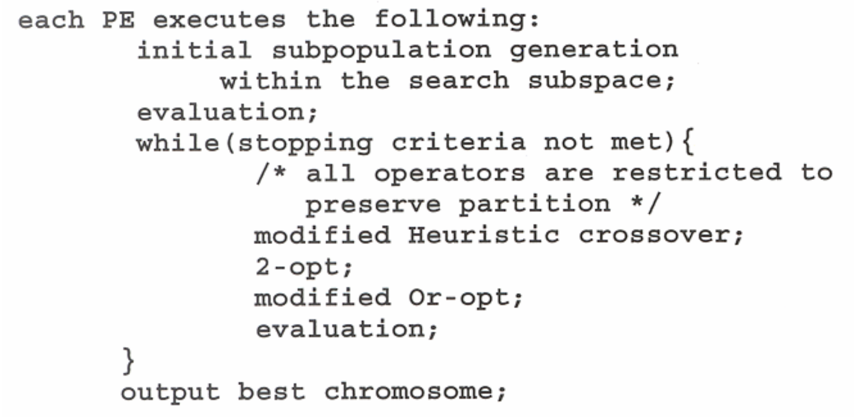
In the migration approach, a balance between subpopulation homogeneity (caused by chromosome migrations) and subpopulation diversity (caused by independent evolutions) must be maintained. If migrations occur too frequently or too many chromosomes are migrated, each PE might end up with the same subpopulation. If migrations occur too infrequently, or too few chromosomes are migrated, the migration approach degenerates to the independent approach. To maintain this balance, the parameters for chromosome migration, such as migration topology, migration size, migration frequency, migrant composition, and replacement policy, must be carefully chosen.

There are a number of existing chromosome migration schemes in the literature. The migration approach in this research used the scheme proposed in [11], which had been tested using a 105-city TSP. In [11], chromosome migrations occur every five iterations. For subpopulation sizes of 50 and 100 chromosomes, the copies of the locally best and second best chromosomes are exchanged at each migration step. Because in this research each PE contains only eight chromosomes (far fewer than 50 and 100), exchanging copies of only the locally best chromosomes (one from each PE) was considered sufficient. Experimental results showed that this approach did perform well. Each PE P sends its migrant to PE $P + 1$ modulo N at the first migration step, then to PE $P + 2$ modulo N at the second migration step, and so on [11]. Mathematically, at the i-th migration step, PE P ($0 \leq P < N$) sends a copy of its best chromosome to

PE (P + 1 + (i modulo (N−1))) modulo N. The expression ‘‘i modulo (N−1)’’ is used to avoid having a PE P send a chromosome to itself. Once a PE receives the migrating chromosome, it checks to see if there is a local chromosome that is identical to the one received. If there is an identical chromosome, the received chromosome is discarded. Otherwise, the received chromosome replaces the worst local chromosome.

5.4 PGA with partitioned search space

A third way of parallelizing GAs is to partition the search space into disjoint subspaces and to force PEs to search in different subspaces. In this research, this is called the partitioned PGA approach, the outline of which is given in Figure 5. The only difference between the partitioned and the independent approaches is that in the partitioned approach each PE searches in a different subspace while in the independent approach all PEs search in the entire search space. The partitioned approach was considered to test whether search space partitioning can reduce PGA execution time while maintaining solution quality.



```
each PE executes the following:
      initial subpopulation generation
          within the search subspace;
      evaluation;
      while(stopping criteria not met){
            /* all operators are restricted to
               preserve partition */
            modified Heuristic crossover;
            2-opt;
            modified Or-opt;
            evaluation;
      }
      output best chromosome;
```

Figure 5. Outline of the partitioned approach.

In this approach, the search space is partitioned by constraining the set of cities that can precede and succeed a given city (city 0 in this research) on the tours for each PE. Each PE is assigned a disjoint set of these cities. Specifically, the search subspace for each PE P ($0 \le P < N$) is defined by a partition matrix π_P of $C-1 \times C-1$ elements, where the rows and columns are numbered from 1 to $C-1$. Each element $\pi_P(m,n)$ ($1 \le m, n < C$) can be either 1 or 0. $\pi_P(m, n) = 1$ means that PE P can have tours that visit city m before city 0 and visit city n after city 0. $\pi_P(m,n) = 0$ prohibits any tours on PE P to have the subtour (m,0, n).

The properties of π_P include

(1) $\pi_P(m, m) = 0$ for any m,

(2) π_P is symmetric, i.e., $\pi_P(m, n) = \pi_P(n, m)$ for any n and m, and

(3) $\sum_{P=0}^{N-1} \pi_P(m,n) = 1$ for any m and n such that $m \neq n$.

These three properties specify the general constraints applied to any tour on any PE. Property (1) does not allow the same city to be both before and after city 0 on any tours. Property (2) ensures that the reversal of a tour appears in the same subspace. This is required because symmetric TSPs are used in this research. Property (3) guarantees that the subspaces are disjoint and the collection of these subspaces form the original full search space.

The initial population generation step is described in Subsection 6.1. To maintain the partitioned search space, some restrictions are imposed on the genetic operators. In each crossover operation, consider the parent tour that has the shorter length three-city subtour with city 0 in the middle. Let m be the first city and n be the third city of this subtour. During the construction of the offspring tour, when city m is chosen and added to the partial offspring tour, cities 0 and n are also added to this partial offspring tour after city m. If city n is chosen before city m and added to the partial offspring tour, cities 0 and m are also added to the offspring tour after city n. City 0 is not allowed be added to the partial offspring tour by any other means. Restrictions are also imposed on each 2-opt and Or-opt operation, where a tour improvement is disallowed if the resulting modified tour does not belong to a PE’s search subspace.

5.5 PGA with tour segmentation and recombination

A novel PGA proposed in this research specifically for the TSP involves tour segmentation and recombination. This is called the segmentation PGA approach. This PGA starts by segmenting tours into subtours. Then after subtour improvements, the subtours are recombined into longer subtours. This subtour improvement and recombination is performed repeatedly until full tours are formed. The execution time of an iteration on subtours is shorter than that on full tours. Iterations on full tours stop when the same stopping criterion used in the other PGAs is met. The purpose of designing the segmentation approach is to test whether by introducing tour segmentation and recombination high quality solutions can be found more quickly than with the other PGAs being compared. Figure 6 shows the outline of the segmentation approach.

```
each PE executes the following:
       initial population and subtour generation;
       evaluation;

       do for each of the first log$_2$N phases
       {      /* all operators are restricted as
                 discussed in Subsection 5.5 */
              modified Heuristic crossover;
              2-opt;
              modified Or-opt;
              evaluation;
              recombination; /* as discussed in
                                 Subsection 5.5*/
       }
       /* the last phase: */
       while(stopping criteria not met){
              modified Heuristic crossover;
              2-opt;
              modified Or-opt;
              evaluation;
      }
      output best chromosome;
```

Figure 6. Outline of the segmentation approach.

At the initial population generation step, a total of $\underline{S}$ tours are generated. Each tour is then evenly segmented into N disjoint subtours, where N is the number of PEs used. PE P is assigned the P-th subtour of each chromosome. Thus, as a result of this tour segmentation, each PE possesses S subtours, each of which is the P-th segment of a full tour.

To correctly calculate the subtour lengths, each PE also knows the first city of the next segment of each full tour. The length of each subtour is thus the travel distance from the first city of the subtour being considered to the first city of the next subtour of the same full tour. The sum of these lengths over all component subtours of a full tour is the full-tour length. The first city of each subtour is not allowed to be changed by genetic operators in order to facilitate the calculation of full-tour lengths. Figure 7 shows an example of the initial local subtour segmentation. (Recall that each PE also knows the first city of the next subtour for each tour.)

Some restrictions must be applied to the Heuristic crossover and local improvement genetic operators to guarantee that the resulting tours are valid. To describe the restriction on Heuristic crossover, the current last city on the partial offspring subtour being constructed will be denoted $\underline{c_{last}}$, and the parent subtours with the longer and the shorter total tour lengths will be denoted $\underline{t_{long}}$ and $\underline{t_{short}}$, respectively. In the segmentation approach, the offspring subtour from Heuristic crossover starts from the first city of t_{long} . This is because the offspring subtour will replace the subtour of t_{long} and thus must start with the same city. If t_{long} contains c_{last} , the city following c_{last} on t_{long} will be called $\underline{c_{nextl}}$. Similarly, if t_{short} contains c_{last} , the city following c_{last} on t_{short} will be called $\underline{c_{nexts}}$. The city c_{nextl} is used to extend the current partial offspring tour if both of the following conditions are true:

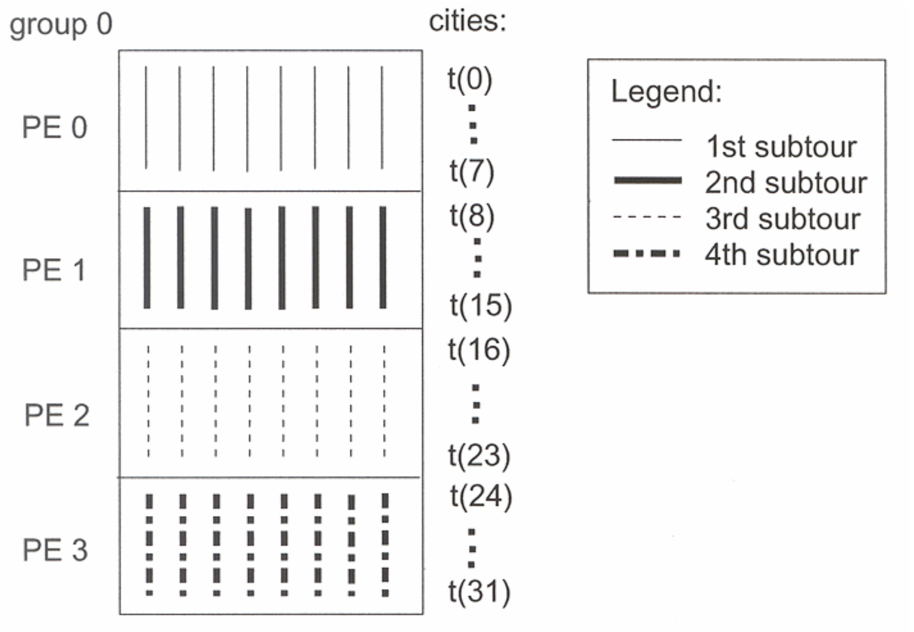


Figure 7. An example of initial local subtour distribution of the segmentation approach (each PE also knows the first city of the next subtour for each tour). In this example, the number of PEs is four, the population size is eight, and the number of cities is 32.

(1) c_{nextl} is not on the current partial offspring tour

(2) c_{nexts} is on the current partial offspring tour, or c_{nexts} is not on t_{long} ,or $d_{c_{last}c_{nextl}} \leq d_{c_{last}c_{nexts}}$

The city c_{nexts} will be used to extend the current partial offspring tour if both of the following conditions are true:

(1) c_{nextl} cannot be used

(2) c_{nexts} is not on the current partial offspring tour and c_{nexts} is on t_{long} .

If neither c_{nextl} nor c_{nexts} can be used, then a city on t_{long}, excluding those that are on the current partial offspring subtour, is selected randomly. In the 2-opt and Or-opt local subtour improvements, whenever a subtour change would result in relocating the first city of the subtour, the change is not performed.

For each chromosome, at the evaluation step, each PE first locally computes the subtour length from this subtour's first city to the first city of the next subtour. Then all the PEs that share a chromosome perform a recursive doubling operation (e.g., [40]). In this operation, each of these PEs provides the lengths of its subtours and gets full lengths of all tours as the outcome of this reduction operation. These full tour lengths are used in the crossover step for deciding which parent tour to be replaced by the offspring tour and in the recombination step described below.

There are n + 1 phases in the segmentation approach. When the PEs sharing tours finish the current phase, they are all involved in a recombination process, as discussed below. First, some notation that will be used for specifying the recombination step is defined. Without loss of generality, it can be assumed that the number of PEs, N, is a power of two. Let $\underline{n} = \log_2 N$. Each PE's physical number P is denoted as $p_{n-1} p_{n-2} \ldots p_1 p_0$. It can also be assumed that the total number of chromosomes (tours), S, is a power of two. Let $\underline{s} = \log_2 S$.

At a given phase e, $0 \leq e \leq n$, 2^{n-e} PEs form a group of PEs that share a set of tours. For phase e, there are 2^e groups, numbered from 0 to $2^e - 1$. Each PE P belongs to group $p_{e-1} \ldots p_1 p_0$ when e > 0 and belongs to group 0 when $e = 0$. At the first phase, when e = 0, all PEs start as a single group sharing all tours, and in the last phase, where $e = n$, each PE is in its own group and shares no tours with other PEs.

The following describes one way of doing the subtour recombination. Each PE contains $S/2^e$ subtours, where each subtour involves $C/2^{n-e}$ cities. Recombination occurs at the end of each phase e, except for the last phase when $e = n$. At the beginning of a recombination, the subtours contained by a group of PEs are numbered in ascending order of the corresponding full-tour lengths, which are known to all PEs in the same group. Therefore, all subtours that belong to the same tour have the same order number across the PEs in a group.

Recombination is performed by having each PE exchange subtours with the PE whose physical number differs only in the *e*-th bit position. In particular, within each group, each PE $p_{n-1} p_{n-2} \ldots p_{e+1} 0\ p_{e-1} \ldots p_0$ sends each corresponding PE $p_{n-1} p_{n-2} \ldots p_{e+1} 1\ p_{e-1} \ldots p_0$ its odd-numbered subtours, and each PE $p_{n-1} p_{n-2} \ldots p_{e+1} 1\ p_{e-1} \ldots p_0$ sends each corresponding PE $p_{n-1} p_{n-2} \ldots p_{e+1} 0\ p_{e-1} \ldots p_0$ its even-numbered subtours. Each PE then concatenates each received subtour to the appropriate end of the locally held subtour with the same tour number. As a result of this chromosome exchange and recombination, each PE now has one half of the subtours it had before, but each subtour is twice as long. Figures 8 and 9 illustrate two recombinations following the initial subtour distribution given in Figure 7.

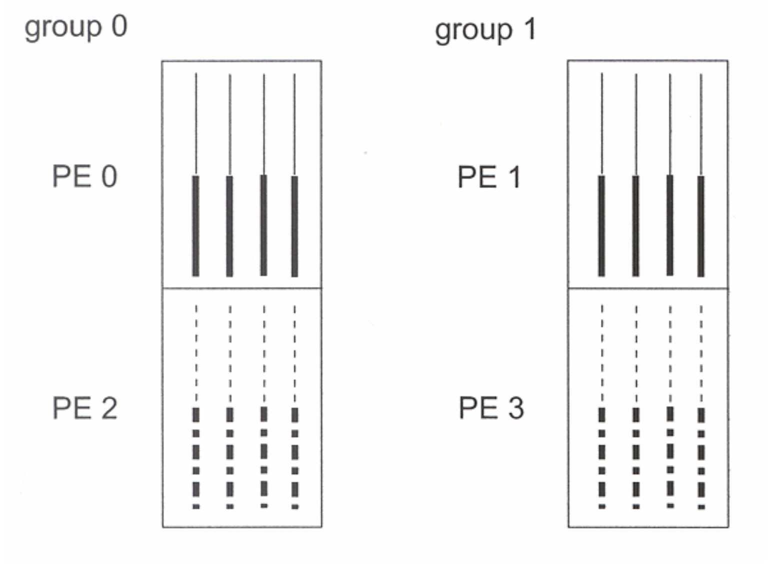


Figure 8. The first recombination in the segmentation approach after Figure 7.

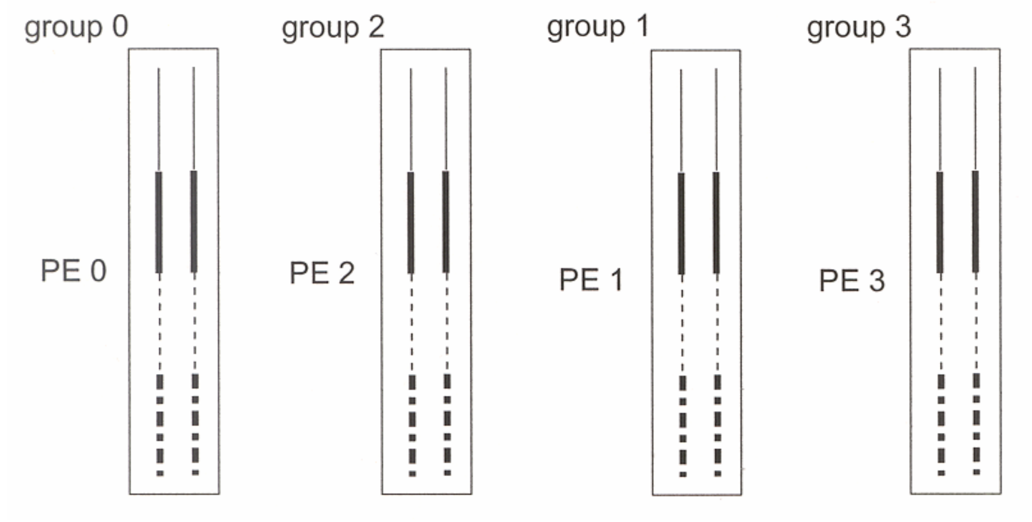


Figure 9. The second recombination after Figure 7.

The subtour numbering by full-tour lengths done within each group of PEs facilitates an approximately equal distribution of quality among the subtours retained after the pairwise inter-PE transfers discussed above. In particular, because PE $p_{n-1} p_{n-2} \ldots p_{e+1} 0\ p_{e-1} \ldots p_0$ gets even-numbered subtours and $p_{n-1} p_{n-2} \ldots p_{e+1} 1\ p_{e-1} \ldots p_0$ gets odd-numbered subtours, the new subtour subpopulations of these two PEs are both of mixed quality.

After a recombination, each PE starts a new phase of the evolution (i.e., *e* is incremented). At each of the first *n* phases, each PE executes the Heuristic crossover, 2-opt, and Or-opt steps on its subtours. The genetic operations in these steps are modified for the segmentation approach as discussed earlier in this subsection. Only one evolution iteration is performed for each of these first *n* phases. At the last phase, when $e = n$, each PE has *S*/*N* chromosomes, each of which is a full tour of *C* cities. At this phase, the segmentation approach becomes the independent approach with (*S*/*N*) full tours assigned to each PE.

The following analysis shows that each iteration on subtours takes less execution time than that on full tours. It is observed that exhaustive Or-opt improvements take the majority of the computation time of each iteration. The execution time spent on exhaustive Or-opt operations on subtours at phase *e* (assuming each operation takes one unit of time) is

$$\underline{T(e)} = \frac{S}{2^e} \times \frac{C}{2^{n-e}} = \frac{SC^2}{N^2} \times 2^e \tag{3}$$

This time doubles each time *e* is incremented. At the last phase, where $e = n$, this time is

$$T(e = n) = \frac{SC^2}{N} \tag{4}$$

It can be seen from the above equation that $T(n) = N \times T(0)$, i.e., the time taken in the last phase is *N* times as expensive as that taken in the first phase. The total local improvement time on tour segments for the first *n* phases (one iteration per phase) is

$$\sum_{e=0}^{n-1} T(e) = \sum_{e=0}^{n-1} \left[\frac{SC^2}{N^2} \times 2^e \right] = \frac{SC^2}{N^2} \times \sum_{e=0}^{n-1} 2^e = \frac{T(n)}{N} \sum_{e=0}^{n-1} 2^e = T(n) \frac{N-1}{N} \tag{5}$$

The above equation reveals that for the segmentation approach, when *N* is large, the sum of the time for all of the first *n* iterations is approximately equal to that of one iteration of the last phase. Also, each of the *n* phases after the first phase is built upon two previously improved component subtours. These two characteristics make it possible for the segmentation approach to find the same or better quality solutions more quickly than those PGAs that always operate on full tours (e.g., the independent approach). However, it is possible that performing the genetic operations in one iteration on full tours may result in generating better new populations. Thus, the experiments in Section 6 are needed to evaluate the trade-offs.

5.6 PGA with tour segmentation, recombination, and migration

The last PGA considered in this research is the segmentation-migration approach, which augments the segmentation approach with periodic chromosome migrations. The motivation for designing this algorithm is to combine the characteristics of both the migration and the segmentation approaches, both of which performed well in the initial experimentation.

The initial subpopulation generation and the first *n* phases of the segmentation-migration approach are exactly the same as in the segmentation approach. However, from the beginning of the last phase, once PEs get subpopulations of full tours, they start periodic chromosome migrations in the same way as those in the migration approach. Specifically, they exchange the best full tours once every five iterations after all PEs are operating on full tours.

6. PGA IMPLEMENTATION DETAILS AND EXPERIMENTAL RESULTS

6.1 PGA initial subpopulation generations

The five PGAs described in this paper (the independent, the migration, the partitioned, the segmentation, and the segmentation-migration approaches) were implemented and tested on the 16 thin PEs of the IBM SP2 parallel machine. The same set of five TSP instances (KA, KC, KD, Reinelt, and lattice TSPs) were used to quantify the performance of these PGAs. To make fair comparisons, the same set of 50 different initial populations of 128 full tours used for testing the baseline SGA were also used for each PGA on each TSP instance. In the independent, the migration, and the partitioned approaches, each initial population was divided into 16 subpopulations, each of which resided on a different PE. As a result, each PE contained eight full tours (which was sufficient to find high quality solutions comparable to those found by the 128-chromosome baseline SGA on a single PE). In each run of the segmentation and the segmentation-migration approaches, each full tour in the initial population was segmented into 16 disjoint subtours. PE P ($0 \leq P < 16$) had 128 subtours as its initial subpopulation, where each of its subtours was the P-th subtour of a full tour.

The partitioned approach required that each subpopulation come from a disjoint search subspace. In the implementation of the partitioned approach, the search space partitioning and the initial subpopulation generation were performed as follows. Each PE constructs its own partition matrix π_P. Consider the portion of π_P comprised of the E elements that were above the main diagonal. Assuming the elements in this portion were ordered in row major format, PE P set to 1 elements $P(E/16)$ to $(P + 1)(E/16) - 1$ in π_P. Then, to make π_P a symmetric matrix, set $\pi_P(n,m)$ to 1 wherever $\pi_P(m,n)$ is 1. All other elements in π_P were set to 0.

After each PE P determines its partition matrix, the initial random population of 128 chromosomes is distributed among the PEs based on the partition matrices. Unless all PEs have exactly eight chromosomes, the PEs exchange portions of the search space (by swapping partition matrix entries with another PE) to evenly distribute the initial chromosomes among all PEs. At the end of this process, the search space is still evenly divided among all of the PEs and each PE contains exactly eight chromosomes from the initial population. While this procedure may not always work in general, it did for all initial populations in this research.

6.2 Experiments to explore limits on solution quality

The relative performance (i.e., solution quality and execution time) of any two PGAs, both starting from one randomly generated initial population, could be different from that of these two PGAs both starting from another randomly generated population. Therefore, to attempt to determine the relative performance, the average values for the solution quality and the execution times over 50 runs for each PGA on each TSP instance were used in the experimental results tables. Two sets of experiments were designed to quantify the performance of these five PGAs. The purpose of the first set of experiments was to measure the quality of the best solution each PGA could find. The purpose of the second set (described in the next subsection) was to determine how quickly each PGA could find comparable solutions of high quality.

The first set of experiments used a stopping criterion of no improvement in the best solution found after 150 iterations. Each run stopped when all PEs achieved this stopping criterion. On each TSP instance, each PGA was run 50 times, each time starting from a different initial population. Table II shows the mean D, the max D, the mean of the execution times, and the standard deviation (σ) of the execution times for each PGA on each TSP instance for this experiment set. In this table and Table VI, confidence intervals [41] of all mean values were calculated. Let X represent either D or the execution time. The mean value of X is expressed in the form of $x \pm \Delta x$, where x is the 50-run mean and Δx was calculated according to Eq. (8.5) in [41] based on the variance of X, the number of runs (50), and a given confidence. In this research, a 90% confidence was used, which means that if another set of runs were conducted (the number of which must be statistically significant (e.g., 50)), there is a 0.9 probability that the sample mean of this second set of runs is in the interval between $x - \Delta x$ and $x + \Delta x$.

From Table II, it can been seen that on all TSP instances the best solutions found by all PGAs, other than the partitioned approach, were within 1% of the best known solutions, the same range of the solution quality found by the baseline SGA. On each TSP instance, the migration approaches took significantly shorter mean time to finish than the non-migration approaches. The segmentation-migration approach, the hybrid of the segmentation and the migration approaches, had the shortest mean execution time. It was also observed that the partitioned approach had the worst performance in both the solution quality and the mean execution time for this set of experiments.

Comparing the SGA and PGA execution times is meaningful because: (1) both the baseline SGA experiments (see Subsection 4) and this set of PGA experiments used the same stopping criterion and (2) the baseline SGA and the PGAs other than the partitioned approach all found comparable quality solutions (all within 1% longer than the best known solutions). The best algorithm identified by this set of PGA experiments, the segmentation-migration approach, was chosen to be compared with the baseline SGA. Define speedup to be the mean execution time of the baseline SGA divided by that of the segmentation-migration PGA approach on a given TSP instance. The average speedup over the five TSP instances was

13.83, which indicates that the parallelization techniques used by the segmentation-migration PGA approach (i.e., chromosome migration, and tour segmentation and recombination) is quite effective in decreasing the execution time while maintaining the solution quality.

The following discussion explains why chromosome migration is effective at reducing the total execution time. In the first set of experiments, the execution time was determined by when the last PE finished. In the migration approach once the globally best solution was found by one PE, after at most five migration steps all PEs had a copy of this chromosome. Because migrations occur once every five iterations, all PEs in the migration approach stop within $5 \times 5 = 25$ iterations of each other. Each PE in the PGAs without chromosome migrations, however, stopped independently. Although some PEs in these PGAs found the globally best solution sooner than other PEs and stopped earlier, some other PEs required a greater number of iterations to improve their local solutions and stopped later. This was why the last PE in the migration approaches always had a shorter mean time to finish in the first set of experiments, compared to the mean time when the last PE in other PGAs finished.

Table II. Performance comparisons of the five PGAs for the first set of experiments, where each PE stopped when the locally best was not improved for 150 iterations.

	D (%) for the five PGAs									
	independent		migration		partitioned		segmentation		segment.-migration	
TSP	μ	max	μ	max	μ	max	μ	max	μ	max
KA	0.00±0.00	0.00	0.00±0.00	0.00	0.12±0.04	0.46	0.00±0.00	0.00	0.00±0.00	0.00
KC	0.00±0.00	0.00	0.00±0.00	0.00	0.77±0.12	1.67	0.01±0.00	0.09	0.00±0.00	0.00
KD	0.03±0.01	0.07	0.01±0.01	0.07	0.64±0.08	1.21	0.02±0.01	0.07	0.03±0.02	0.45
RE	0.01±0.01	0.43	0.00±0.00	0.08	0.69±0.09	1.67	0.01±0.00	0.09	0.01±0.01	0.43
LT	0.38±0.04	0.83	0.04±0.01	0.83	1.13±0.11	1.66	0.16±0.02	0.83	0.16±0.01	0.83
	execution time (sec) for the five PGAs									
	independent		migration		partitioned		segmentation		segment.-migration	
TSP	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
KA	42.04±1.45	6.12	28.24±0.41	1.76	53.93±1.73	7.28	43.35±1.43	6.03	26.89±0.42	1.79
KC	45.92±1.37	5.78	32.42±0.98	4.13	54.81±2.07	8.75	42.50±1.09	4.60	29.37±0.77	3.23
KD	48.70±2.03	8.56	35.80±1.00	4.23	60.38±2.15	9.07	48.76±1.50	6.33	30.97±0.90	3.82
RE	54.08±1.68	7.07	38.33±1.52	6.42	59.28±2.18	9.19	53.11±1.67	7.02	37.92±2.33	9.82
LT	39.59±1.16	4.90	31.65±1.45	6.14	44.76±1.17	4.95	39.93±1.56	6.60	30.04±1.14	4.82

For example, from the same initial population on the KA TSP, both the migration and the segmentation approaches found the best known solution. However, these two runs found the best solution at different iterations and stopped at different iterations. Table III shows when the first PE found the best known solution and when the last PE stopped for these two approaches. (Recall that the stopping criterion was 150 iterations with no improvement of solution quality.) Table IV provides more detailed information about the stopping behavior of the migration approach. Each row in Table IV specifies the number of PEs that found the best known solution independently and the number of PEs that received the best solution from another PE at the iteration shown in the first column. The total number of PEs that obtained the best solution is also shown.

From Table III, it can be seen that despite the fact that one of the PEs in the segmentation approach found the globally best solution faster than any PE in the migration approach (18th iteration versus the 22nd iteration), it took longer for the segmentation approach to stop (264th iteration versus the 190th iteration). This is due to the fact that the segmentation approach requires each PE to independently determine the globally optimal solution whereas the migration approach communicates the best solution to PEs that may be searching in undesirable portions of the search space. For the example in Table IV, eleven of the sixteen PEs received the globally best solution from another PE.

Table II also shows that compared with other PGAs, the partitioned approach had the worst performance in terms of both solution quality and mean execution time on every TSP instance. The following example runs demonstrate the differences between the partitioned and the independent approaches.

Table III. The iteration and finish time when the first PE found the best known solution and when the last PE stopped in the runs of the migration and the segmentation migration approaches.

	iteration		time (sec)	
	migration	segmentation	migration	segmentation
when first PE found best	22	18	4.17	3.56
when last PE stopped	190	264	26.10	34.17

All runs on the KA TSP in this research revealed that there was only one tour that had the best known tour length. Consider two typical runs from the first set of experiments, one using the independent approach and the other using the partitioned approach. They both started from the same initial population on the KA TSP. It was found that the best known solution was in PE 11's search subspace (the optimal subspace) for the partitioned approach.

Table IV. A typical run of the migration approach.

	#PEs found global best independently		#PEs received global best from another PE		total #PEs found globally best	
iteration	this iteration	so far	this iteration	so far	this iteration	so far
22	1	1	0	0	1	1
25	1	2	1	1	2	3
26	1	3	0	1	1	4
30	1	4	4	5	5	9
34	1	5	0	5	1	10
35	0	5	4	9	4	14
40	0	5	2	11	2	16

In the partitioned approach, when the best known solution was within one PE's subspace (which was the case for this example), there was only a single PE (PE 11 in the example considered) that was searching in the optimal subspace. All other PEs were searching in subspaces that did not contain the best known solution. In this specific example, the best known solution was not found. The best solution found (by PE 11) had a value of D = 0.46, which was the worst for all PGA runs conducted on the KA TSP in the first set of experiments.

When the independent run stopped, it was found that all chromosomes in fourteen of the 16 subpopulations were located in a region of the search space that corresponded to a single PE's partition (that of PE 11 in the example above). That is, eventually, there were fourteen PEs that were searching in the optimal subspace. Of these fourteen PEs, seven of them found the best known solution. (The worst of these fourteen final solutions had a value of D = 1.37, larger than that of the best solution found in the partitioned run (0.46)). Thus, approaches that do not force disjoint subpopulations among PEs allow multiple PEs to search in subpopulations that have more potential for producing a good solution. Furthermore, the partitioned approach requires more work per iteration due to the validity checks to enforce disjoint subpopulations.

For this set of experiments, the segmentation-migration approach found solutions all within 1% of the best known solutions with the smallest mean execution time on each TSP instance. To compare the mean execution times of different PGA approaches when they achieve virtually identical quality solutions, a second set of experiments was conducted and is described next.

6.3 Experiments to explore relative execution times

The second set of experiments were designed to find out how quickly different PGAs could find solutions of the same high quality. Because the partitioned approach performed poorly compared with other PGAs in the first set of experiments, it was not evaluated using this set of experiments. The same set of 50 initial populations was used for each PGA on each TSP instance. To make fair comparisons of the mean execution times, the longest final solution found by a PGA (other than the partitioned approach) for each TSP instance in the first set of experiments was chosen as the basis for the stopping criterion. Each run in the second set of experiments stopped when any PE found a tour that was no worse than this chosen solution. Table V shows the stopping criteria for this set of experiments, which were taken from Table II. The solutions used as the stopping criteria were considered of high quality because each of them was within 1% longer than the best known solution.

Table VI lists the mean of the execution times, the 90% confidence interval for the mean, and the standard deviation of the execution times for each PGA on each TSP instance for the second set of experiments. From this table it can be seen that the migration approaches achieved the shortest mean execution time for each of the TSP instances. Specifically, the migration approach was the best for two of the five TSP instances (KA TSP and LT TSP) and the the migration-segmentation approach was the best for the other three TSP instances. For each TSP instance, the independent approach always had a longer mean execution time than the migration approach, and the segmentation approach always had a longer mean execution time than the segmentation-migration approach.

This set used a stopping criterion that may not be practical in some situations because solutions chosen as the stopping criteria may not be available. However, this set of experiments provided insights into each PGA's inherent ability to quickly find high quality solutions.

Table V. Quality of the solutions taken as the stopping criteria for the second set of experiments.

	TSP				
	KA	KC	KD	RE	LT
D (%)	0.00	0.09	0.45	0.43	0.83

Table VI. Execution time performance of non-partitioned PGAs for the second set of experiments, where each PE stopped when then locally best solution was at least as good as the given solution.

	execution time (sec) for four PGAs							
	independent		migration		segmentation		segmentation-migration	
TSP	μ	σ	μ	σ	μ	σ	μ	σ
KA	6.74±0.70	2.95	4.82±0.37	1.55	6.52±0.61	2.58	5.06±0.41	1.74
KC	8.74±1.51	6.36	6.63±0.92	3.90	7.83±1.13	4.77	5.75±0.72	3.03
KD	5.97±0.65	2.76	5.25±0.45	1.90	5.43±0.62	2.60	4.46±0.42	1.76
RE	8.68±1.74	7.35	6.83±0.86	3.61	7.29±1.02	4.30	6.03±0.85	3.57
LT	4.41±0.71	2.99	4.04±0.58	2.43	5.62±1.17	4.94	4.32±0.60	2.52

Different from the first set of experiments, the second set of experiments had larger standard deviations of the execution times relative to the mean values. This is because each run in the first set of experiments stopped when the last PE did not have its best solution improved for 150 iterations. In general, these 150 iterations had a non-negligible impact on the total execution times of the runs of each PGA on each TSP instance (e.g., see Table III). This resulted in a relatively more homogeneous set of execution times than if these extra iterations were not included in the execution times. The second set of experiments, without this time for the 150 iterations, had relatively larger standard deviations.

7. CONCLUSIONS

Four conceptually different PGA approaches and one hybrid of two of these approaches were compared using the TSP as an example application problem. For fair comparisons, all PGAs were developed based on the same baseline SGA, implemented on the same 16 thin PEs of an IBM SP2 parallel machine, and tested using the same set of initial populations on the same set of TSP instances. Two techniques were identified as the most effective ones for improving the PGA performance: (1) inter-PE information exchange during the evolution progress (e.g., migration of the locally best chromosomes) and (2) tour segmentation and recombination. The segmentation-migration PGA approach, which combined the above two techniques, took the shortest execution time in the first set of experiments (where a practical stopping criterion was used). It was shown in the second set of experiments that the migration and the segmentation-migration approaches were able to find solutions of similar high quality faster than the other approaches.

There are a great number of other possible variations of these five PGA approaches that can be created by varying PGA design parameters that include but are not restricted to the following: migration parameters (e.g., migration partner, interval, and size; migrant selection; and replacement policy), selection mechanism, whether elitism is enforced, crossover and mutation operators, probabilities for performing genetic operators, stopping criteria, population size, application problem, size of the application problem, number of PEs, and type of parallel machine. In this work, based on a combination of information available in the literature and experiments conducted, values for these design parameters were chosen and, to the extent relevant, held constant across the PGA approaches tested. Certainly, future research can build on the results presented here by considering variations on these PGA approaches, by comparing conceptually new PGAs to these approaches, and by examining other application problems.

ACKNOWLEDGMENTS

The authors thank Prasanna Jog for his help on implementation details of the baseline SGA. Muthucumaru Maheswaran is also thanked for his comments and assistance. An earlier version of this manuscript was presented at the *12th International Parallel Processing Symposium*. This work was supported in part by Architecture Technology Corporation under contract number 6005.

REFERENCES

1. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, *Special Issue on Parallel Evolutionary Computing,* Vol. 47, No. 1, pp. 8-22, 1997.
2. J. T. Alander, *An Indexed Bibliography of Distributed Genetic Algorithms*, Technical Report No. 94-1-PARA, Department of Information Technology and Production Economics, University of Vaasa, Finland, 1997.
3. A. Chipperfield and P. Fleming, "Parallel genetic algorithms," in *Handbook of Parallel and Distributed Computing*, A. Y. Zomaya, Ed. New York, NY: McGraw-Hill, pp. 1118-1143, 1996.
4. M. Gorges-Schleuter, "ASPARAGOS: An asynchronous parallel genetic optimization strategy," *3rd Int'l Conf. Genetic Algorithms*, pp. 422-427, 1989.
5. R. A. Henry, N. S. Flann, and D. W. Watson, "A massively parallel SIMD algorithm for combinatorial optimization," *1996 Int'l Conf. Parallel Processing*, Vol. II, pp. 46-49, 1996.
6. V. Kommu and I. Pomeranz, "Effect of communication in a parallel genetic algorithm," *1992 Int'l Conf. Parallel Processing*, Vol. III, pp. 310-317, 1992.
7. J. Lienig, "A parallel genetic algorithm for performance-driven VLSI routing," *IEEE Trans. Evolutionary Computation*, Vol. 1, No. 1, pp. 29-39, 1997.
8. H. Muhlenbein, "Parallel genetic algorithms, population, genetics and combinatorial optimization," *Third Int'l Conf. Genetic Algorithms*, pp. 416-421, 1989.
9. N. Neves, A.-T. Nguyen, and E. L. Torres, "A study of a non-linear optimization problem using a distributed genetic algorithm," *1996 Int'l Conf. Parallel Processing*, Vol. II, pp. 29-36, 1992.
10. C. C. Pettey, M. Leuze, and J. Grefenstette, "A parallel genetic algorithm," *Second Int' Conf. Genetic Algorithms*, pp. 155-164, 1987.
11. T. Starkweather, D. Whitley, and K. Mathias, "Optimization using distributed genetic algorithms," in *Parallel Problem Solving from Nature*, H. P. Schwefel and R. Manner, Eds. New York, NY: Springer-Verlag, pp. 176-183, 1992.
12. R. Tanese, "Parallel genetic algorithm for a hypercube," *Second Int'l Conf. Genetic Algorithms*, pp. 177-183, 1987.
13. R. Tanese, "Distributed genetic algorithms," *3rd Int'l Conf. Genetic Algorithms*, pp. 434-439, 1989.
14. E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis,* Vol. 10, No. 2, pp. 141-171, 1998.
15. E. Cantú-Paz, "Markov chain models of parallel genetic algorithms," *IEEE Transactions on Evolutionary Computing,* Vol. 4, No. 3, pp. 216-226, 2000.
16. G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," *Evolutionary Computing*, Vol. 7, No. 3, pp. 7-12, 1999.

17. E. L. Lawler, J. K. Lenstra, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. New York, NY: Wiley-Interscience, 1985.
18. G. Reinelt, "TSPLIB -A traveling salesman problem library," *ORSA Journal on Computing*, Vol. 3, No. 4, pp. 376-384, 1991.
19. T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and D. M. Snir, "SP2 system architecture," *IBM Systems Journal*, Vol. 34, No. 2, pp. 152-184, 1995.
20. C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D. G. Grice, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker, "The SP2 high-performance switch," *IBM Systems Journal*, Vol. 34, No. 2, pp. 185-204, 1995.
21. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA: The MIT Press, 1994.
22. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. Cambridge, MA: The MIT Press, 1996.
23. R. Brady, "Optimization strategies gleaned from biological evolution," *Nature*, Vol. 317, No. 6040, pp. 804-807, 1985.
24. J. Dzubera and D. Whitley, "Advanced correlation analysis of operators for the traveling salesman problem," in *Parallel Problem Solving from Nature - PPSN III*, H. P. Schwefel and R. Manner, Eds. New York, NY: Springer-Verlag, pp. 68-77, 1994.
25. B. Freisleben and P. Mertz, "New genetic local search operators for the traveling salesman problem," in *Parallel Problem Solving from Nature - PPSN IV*, H. P. Schwefel and R. Manner, Eds. New York, NY: Springer-Verlag, pp. 890-899, 1996.
26. D. E. Goldberg and R. Lingle, "Alleles, loci, and the traveling salesman problem," *1st Int'l Conf. Genetic Algorithms*, pp. 154-159, 1985.
27. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading MA: Addison-Wesley, 1989.
28. J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. Los Altos, CA: Morgan Kaufmann, 1987.
29. J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," *1st Int'l Conf. Genetic Algorithms*, pp. 160-168, 1985.
30. P. Jog, J. Y. Suh, and D. Van Gucht, "The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem," *3rd Int'l Conf. Genetic Algorithms*, pp. 110-115, 1989.
31. G. Liepins and H. Hilliard, "Greedy genetics," *2nd Int'l Conf. Genetic Algorithms*, pp. 90-99, 1987.
32. H. Muhlenbein, M. Gorges-Schleuter, and O. Kramer, "Evolution algorithms in combinatorial optimization," *Parallel Computing*, Vol. 7, No. 4, pp. 269-279, 1987.
33. I. Oliver, D. Smith, and J. Holland, "A study of permutation crossover operators on the traveling salesman problem," *2nd Int'l Conf. Genetic Algorithms*, pp. 224-230, 1987.
34. D. Sirag and P. Weisser, "Toward a unified thermodynamic genetic operator," *2nd Int'l Conf. Genetic Algorithms*, pp. 100-107, 1987.
35. A. Y-C. Tang and K-S Leung, "A modified edge recombination operator for the traveling salesman problem," in *Parallel Problem Solving from Nature - PPSN III*, H. P. Schwefel and R. Manner, Eds. New York, NY: Springer-Verlag, pp. 180-188, 1994
36. P. Moscato and M. G. Norman, "A memetic approach for the traveling salesman problem: Implementation of a computational ecology for combinatorial optimization on message-passing systems," *Parallel Computing and Transputer Applications*, M. Valero, E. Onate, M. Jane, J.L. Larriba and B. Suarez, Ed. Amsterdam: IOS Press, pp. 187-194, 1992.
37. N. J. Radcliffe and P. D. Surry, "Formal Memetic Algorithms," *Selected Papers from AISB Workshop on Evolutionary Computing*, London: Springer-Verlag, pp. 1-16, 1994.
38. G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Trans. Neural Networks*, Vol. 5, No. 1, pp. 96-101, 1994.
39. L. Wang, *A Genetic-Algorithm-Based Approach for Task Matching and Scheduling in Heterogeneous Computing Environments and a Comparative Study on Parallel Genetic Algorithms*, Ph.D. Thesis, School of Electrical and Computer Engineering, Purdue University, 1997.

40. H. J. Siegel, L. Wang, J. J. E. So, and M. Maheswaran, "Data parallel algorithms," in *Handbook of Parallel and Distributed Computing*, A. Y. Zomaya, Ed. New York, NY: McGraw-Hill, pp. 466-499, 1996.
41. A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. New York, NY: McGraw-Hill, 1982.

ABOUT THE AUTHORS

L. Wang received his BSEE from Tsinghua University, Beijing, China, and Ph.D. in electrical and computer engineering from Purdue University, West Lafayette, Indiana, USA, in 1990 and 1997, respectively. He joined Microsoft Corporation in Redmond, Washington, USA, in 1997. His research interests include heterogeneous computing, genetic algorithms, and data mining. Dr. Wang has authored journal and conference papers, book chapters, and has several patents pending.

A. A. Maciejewski received the BSEE, MS, and PhD degrees from Ohio State University in 1982, 1984, and 1987. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at Purdue University, West Lafayette. He is currently the Department Head of Electrical and Computer Engineering at Colorado State University. Homepage: www.engr.colostate.edu/~aam.

H. J. Siegel is the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering, and a Professor of Computer Science, at Colorado State University. From 1976 to 2001, he was a professor at Purdue University. He is an IEEE Fellow and an ACM Fellow. Home page: www.engr.colostate.edu/~hj.

V. P. Roychowdhury received the Ph.D. in electrical engineering from Stanford University in 1989. From 1991 to 1996, he was a faculty member with the School of Electrical and Computer Engineering, Purdue University, where he was promoted to Associate Professor in 1995. In 1996, he joined the University of California, Los Angeles, where he is currently a Professor of electrical engineering. He also serves on the faculty of the Biomedical Engineering Interdepartmental Program.

B. Eldridge is currently working on his B.S. degree in electrical engineering at Colorado State University. His current research interests are robotics, control systems, human-robot interaction, and genetic algorithms.