Journal of Supercomputing

GPU-based Parallel Genetic Approach to Large-scale Travelling Salesman Problem

--Manuscript Draft--

Manuscript Number:	SUPE-D-15-00673
Full Title:	GPU-based Parallel Genetic Approach to Large-scale Travelling Salesman Problem
Article Type:	S.I. : HPC in Computational Intelligence, Machine Learning and Informatics
Keywords:	Travelling salesman problem; Genetic algorithms; Constructive crossover; GPU parallelism.
Corresponding Author:	Young-Min Kang, Ph.D. Tongmyong University Busan, KOREA, REPUBLIC OF
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Tongmyong University
Corresponding Author's Secondary Institution:	
First Author:	Young-Min Kang, Ph.D.
First Author Secondary Information:	
Order of Authors:	Young-Min Kang, Ph.D.
	Semin Kang, Master
	Sung-Soo Kim, Master
	Jong-Ho Won
Order of Authors Secondary Information:	
Funding Information:	Electronics and Telecommunications Research Institute (KR) (15ZS1400) — Prof. Dr. Young-Min Kang
	Korea Institute of Science and Technology Information (KR) (PLSI) — Prof. Dr. Young-Min Kang
Abstract:	Travelling salesman problem (TSP) is a well-known NP-hard prob- lem. Therefore, it is difficult to efficiently find the solution of TSP with the large number of instances. Evolutionary approaches such as genetic algorithm have been widely applied to explore the huge search space of TSP. However, the feasibility constraints of TSP make it difficult to devise an effective crossover method. In this paper, we propose an improved constructive crossover for TSP. As the performance of graphics processing units (GPUs) is rapidly improved, GPU-based acceleration is generally required for computation-intensive prob- lems. However, unfortunately, the constructive crossover methods cannot be easily implemented in a parallel fashion because the each gene element of off- springs is dependent on the previous element in the gene string. In this paper, we propose an effective method with which large number of genes are efficiently evolved by exploiting the parallel computing power of GPUs.

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

Click here to download Manuscript JSC2015_Kang.tex

Noname manuscript No.
(will be inserted by the editor)

GPU-based Parallel Genetic Approach to Large-scale Travelling Salesman Problem

Semin Kang · Sung-Soo Kim · Jong-Ho Won · Young-Min Kang

Received: date / Accepted: date

Abstract Travelling salesman problem (TSP) is a well-known NP-hard problem. Therefore, it is difficult to efficiently find the solution of TSP with the large number of instances. Evolutionary approaches such as genetic algorithm have been widely applied to explore the huge search space of TSP. However, the feasibility constraints of TSP make it difficult to devise an effective crossover method. In this paper, we propose an improved constructive crossover for TSP. As the performance of graphics processing units (GPUs) is rapidly improved, GPU-based acceleration is generally required for computation-intensive problems. However, unfortunately, the constructive crossover methods cannot be easily implemented in a parallel fashion because the each gene element of offsprings is dependent on the previous element in the gene string. In this paper, we propose an effective method with which large number of genes are efficiently evolved by exploiting the parallel computing power of GPUs.

This work was supported in part by ETRI R&D Program (Development of Big Data Platform for Dual Mode Batch-Query Analytics, 15ZS1400) and in part by 2015 KISTI PLSI Program.

Semin Kang
Department of Computer Media Engineering, Tongmyong University, Busan, Korea.
E-mail: semins@tu.ac.kr

Sung-Soo Kim
Eletronics and Telecommunication Research Institute, Daejeon, Korea.
E-mail: sungsoo@etri.re.kr

Jong-Ho Won
Eletronics and Telecommunication Research Institute, Daejeon, Korea.
E-mail: jhwon@etri.re.kr

Young-Min Kang
Department of Game Engineering, Tongmyong University, Busan, Korea.
Tel.: +82-51-629-1253
Fax: +82-51-629-1249
E-mail: ymkang@tu.ac.kr

Keywords Travelling salesman problem, Genetic algorithms, Constructive crossover, GPU parallelism

1 Introduction

Travelling salesman problem (TSP) is a well-known optimization problem where the objective is to find the lowest cost of a tour path that passes all the nodes exactly once and returns back to the starting node. Although it seems simple, this optimization problem is known as 'NP-complete.' In other words, TSP has a huge search space, and it is very hard to find the optimal solution [12,14]. Therefore, various kinds of genetic algorithms have been employed to solve this problem [10,5].

Evolutionary method is a search algorithm based on the rules of evolution like natural selection and natural genetics [9,3]. Briefly described, genetic algorithms employ selection, crossover, and mutation to improve the solutions expressed as genes. In this process, the crossover plays the most important role in producing better genes out of the current gene pool. However, the traditional crossover methods cannot be used for evolutionary approaches to TSP because simple exchange of substrings of genes easily violates the feasibility constraints. In order to avoid this problem, researchers have proposed various crossover methods which always generate feasible offsprings [8,4,15].

Among those crossover methods, 'sequential constructive crossover (SCX) showed the best convergence compared to the previous methods [2]. There are a few disadvantages when the SCX is employed. First, the SCX searches only in forward direction and does not take into account the circular properties of TSP tours. Another disadvantage of the SCX is that the offsprings are likely to be similar to the better one between two parents. This is inevitable because the construction process of the SCX is greedy. The greedy aspect of the SCX makes the gene pool rapidly converge to local minima and reduce the diversity in the gene pool. In order to improve the performance of the constructive crossover, bidirectional circular SCX (BCSCX) was proposed [13]. Although the BCSCX improves the convergence speed, it still suffers from rapid assimilation of genes to a certain local minima.

In this paper, we propose an improved crossover that maintains the diversity of genes. This method produces offsprings which equally inherits from two parents. In this method, the parents alternatingly play their roles in determining the city sequence of offsprings so that it is named 'alternating recommendation crossover (ARX).

The evolutionary methods search better when the gene population is large and genes are diverse. Therefore, it is necessary to use a large amount of genes [7], the computation tasks applied to genes are equivalent. This is a typical 'data parallelism, and GPUs are suitable for such problems. However, the constructive crossovers such SCX, BCSCX, and ARX cannot be easily performed in a parallel fashion. In this paper, we propose an efficient parallel

computing techniques for the constructive crossovers in order to solve large-scale TSPs in an efficient way.

2 Evolutionary approach to travelling salesman problem

In this section, the crossover methods suitable for TSP are introduced. The crossover methods satisfy the constraints of TSP and the offsprings are unconditionally feasible.

2.1 Gene representation and crossover for feasible offsprings

A TSP solver based on genetic algorithm requires a proper gene representation for feasible solutions. The sequence of cities in accordance with the feasible tour can be used as a gene representation.

The solutions must contain all nodes to be visited only once and there must be no missing nodes. In conventional crossover methods, an offspring is generated by exchanging the some parts of chromosomes of parents. However, such a conventional crossover may easily produce duplicate nodes and missing nodes.

Crossover operators such as 'edge recombination crossover (ERX)', 'generalized n-point crossover (GNX)' and 'sequential constructive crossover (SCX)' have been proposed in order to guarantee the feasibility of offsprings, the SCX showed the better fitness convergence than other methods. The SCX crossover method guarantees the validity of offsprings' chromosomes and conserves the merits of parents. This method tries to reduce the local distance between the adjacent nodes in the offspring by sequentially scan the chromosomes of parents. The algorithm can be described as follows[2]:

1. Set the starting node 0 to be the current city p.
2. Find the two unvisited node a and b respectively from the chromosome of each parent by sequentially search the first unvisited node (legitimate node) after the current city p. If the search failed, select any unvisited node from the city template permutation such as <1, 2, ..., n>.
3. Compare the distances from p to a (d_{pa}) and to b (d_{pb}). If d_{pa} is less than d_{pb}, add a to the offspring chromosome and set a to be the current node. Otherwise, b is added and set to be the current node. Then go to step 2.

2.2 Improved SCX with bidirectional and circular search

In order to improve the performance of SCX, bidirectional circular SCX (BC-SCX) was proposed. This method can search the next possible 'legitimate' nodes in the chromosomes of parents in two directions and the chromosome is regarded as a circular data with no ends.

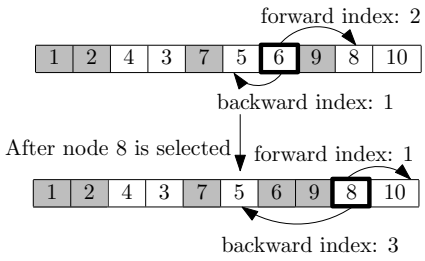


Fig. 1 Example of jump indices and their modification

The BCSCX operator searches legitimate nodes in two directions. In other words, it chooses the two candidate nodes to be added to the chromosome of offspring both before and after the currently visited node from chromosome of each parent.

For example, assume that a uncompleted chromosome sequence <1,5> has been inherited from two parents during the crossover operation. Then the current node is city 5. If, within the chromosome sequence of one parent, the city 3 is the closest unvisited node ('closest' in the aspect of the location in the sequence string) among the nodes after the current node (the city 5), and city 2 is the closest unvisited node before the current node in the sequence of one parent. The proposed BCSCX takes both nodes (cities 3 and 2) as 'legitimate candidate nodes.' Two more candidates are taken from the other parent in the same way. Let us assume that the candidates from the other parent are city 2 and city 6. The candidates for the next node in the offspring's chromosome right after the city 5 are the union of the candidate sets (i.e., cities 2, 3, and 6), and they are tested similarly as SCX. In this method, offspring is constructed in a greedy way so that the offspring is very similar to the better gene when the gap between the fitness values of the parents is huge.

SCX does not asume that chromosome strings are circulary concatenated. Therefore, if the last node of the chromosome string is the current node p, no candidate legitimate nodes can be obtained. In order to avoid this problem, SCX employed a pre-defined template.

For example, assume that the chromosome of a parent is <1,3,7,6,2,4,5> and that of the other is <1,5,7,2,6,3,4>, and currently constructed partial chromosome of the offspring is <1,5>. The original SCX then fails to find the legitimate node from the first parent, and the first unvisited node (in this case, city 2) from the template <1,2,3,4,5,6,7> will be selected as the legitimate node. However, our method regards the chromosomes as circular data, and jumps to the first character so that the city 3 will be selected as the legitimate node.

2.3 Efficient search for legitimate nodes

The constructive crossovers such as SCX construct the chromosomes of offsprings by selecting the best node from the 'legitimate' candidates, and the

Table 1 The Initial setting of parent chromosome before bidirectional constructive crossorver

i	0	1	2	3	4	$\cdots$	$n-1$
χ_i	1	arbitrary city permutation with (2,3,..., n)					
f_i^χ	1	1	1	1	1	$\cdots$	2
b_i^χ	1	2	1	1	1	$\cdots$	1
visited	Yes	No	No	No	No	$\cdots$	No

feasibility of the offspring chromosome is guaranteed by the 'legitimacy' of the candidate nodes. However, the performance of the crossover largely depends on how to search the legitimate nodes. Moreover, the method proposed in this paper searches the legitimate nodes in two directions, and the performance of this search process affects the overall performance of the system.

The overall performance of evolutionary TSP solver depends on three major factors: 1) k, the number of iterations needed for convergence to a reasonable solution, 2) m, the population of genes, and 3) n, the number of cities. If we denote the cost of the legitimate node search for constructing a child chromosome as $search(n)$, the performance of the system can be expressed as $O(km \cdot search(n))$. When a naïve approach such as sequential search is applied, it is obvious that $search(n)$ is $O(n^2)$ and the overall performance will be $O(kmn^2)$.

Since the genes converge to similar genes as the number of iterations increases, the backward search used in the method proposed in this paper inevitably requires $O(n^2)$ searches for the construction of one chromosome.

In order to resolve this problem, we employed forward and backward jump indices. The jump indices can be described as Fig.1. In this figure, the visited nodes are shaded and the most recently visited node has a thick border. The legitimate nodes are searched from this node, and the indices make it possible to search them in $O(1)$ time.

If we consider the first chromosome state, the first five cities of offspring chromosome have been determined, and the last city is 6. The candidates for the next city are city 8 and city 5. Therefore, the forward jump index is 2, and the backward one is 1. If the city 8 is selected as the next city, the jump indices have to be updated. The forward and backward indices of city 8, for example, become 1 and 3.

Because the solution routes of TSP must return to the starting node, we assumed all the feasible chromosome starts from city 1. Let us denote the chromosome satisfying this constraints as χ, and the i-th city in the chromosome as χ_i. The forward and backward jump indices to find legitimate nodes from χ_i are denoted f_i^χ and b_i^χ respectively. The information in the chromosome of each parent before the construction of child choromosome can be initialized as shown in Table 1.

Based on the circular property of the TSP solution, the indices restarts from 0 when it becomes larger than $n-1$ (i.e., i mod n). Similarly, the indices comes down from $n-1$ when it becomes less than 0. Let us denote index

i satisfaying this constraints as $< i >_n$. The node 0 (i.e., χ_0) is alway 1. The forward and backward indices of all nodes are initialized as 1 except for the forward one of node $n-1$, and backward one of node 1 because χ_0 will be automatically inherited to child's choromosome and regarded an already visited node. Because BCSCX takes four candidate legitimate nodes from two parent choromosome, there is no guarantee that χ_1 or χ_{n-1} will be selected as the next node. If a node χ_i is selected as the next visiting city, only two indices $f^{\chi}_{<i-b^{\chi}_i>_n}$ and $b^{\chi}_{<i+f^{\chi}_i>_n}$ in the chromosome must be updated. The update can be done as follows:

$$\begin{aligned} f^{\chi}_{<i-b^{\chi}_i>_n} &= f^{\chi}_{<i-b^{\chi}_i>_n} + f^{\chi}_i \\ b^{\chi}_{<i+f^{\chi}_i>_n} &= b^{\chi}_{<i+f^{\chi}_i>_n} + b^{\chi}_i \end{aligned} \tag{1}$$

With the assistance of the indices managed as shown in Eq. 1, the total search for legitimate nodes during the whole construction of child chromosome can be done in $O(n)$. Therefore, the overall system performance is $O(kmn)$, and we have only to improve the convergence speed to reduce k when devising a better constructive crossover method.

2.4 Alternating recommendation crossover: ARX

The most important drawback of SCX and BCSCX is that an offspring is biased to a better parent. This is inevitable because of the greedy aspect of SCX and BCSCX in the construction of offspring genes by selecting the best unvisited node at every step. The greedy property makes the gene pool rapidly lose the diversity, and the evolution process easily become stagnant at local minima.

In order to avoid this problem, we propose a method that maintains the diversity of the genes by alleviating the greediness of crossover and making offsprings inherit the parents with the equivalent importance. The proposed crossover was named 'alternating recommendation crossover (ARX) because the parents alternatingly take privilege to determine the gene sequence of offsprings. Each parent can recommend the legitimate nodes as in the BCSCX only when it is given the privilege. Therefore, the parents equally play roles in constructing the offspring gene sequence.

The actual crossover method is shown in Fig. 2. Two parents are denoted α and β, and two points where the recommendation privilege is switched are determined. The crossover produces two offsprings (child α and child β). In the construction of one offspring, parent α recommends the legitimate nodes until the first crossover point A, and parent β takes over the privilege after the point. At the next crossover point marked as B, the privilege is again switched to parent α. In the construction of child β, the recommendation is performed in the reverse manner. As shown in the figure, ARX can produce two different offsprings while SCX and BCSCX can produce only one offspring

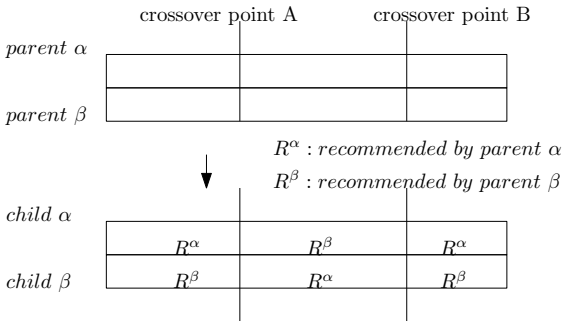


Fig. 2 Offspring construction of ARX

at each crossover. The substrings recommended by parent α and β are denoted R^α and R^β respectively.

The ARX can maintain the diversity of genes compared to SCX and BC-SCX. The fitness improvement of ARX is not rapid compared to other constructive crossover methods because it equally takes into account the features of worse parent. However, the diversity of gene pool enables the evolution to escape from the local minima more easily than the previous constructive methods, and to find the better solution in the long term.

3 GPU acceleration

The graphics processing units (GPUs) have been intially devised to transform a large amount of geometric data. Therefore, GPUs are parallel processors that perform simple and similar tasks to large data, and they are suitable for various 'data parallel' problems [6]. As the performance improvement of GPUs outperform traditional CPUs, various computation problems such as molecular dynamics are being successfully accelerated with GPUs [16]. Moreover, GPU resources are relatively cheaper than CPUs so that more and more high performance computing prblems are accepting GPU-based parallel computing [17].

However, there are some limitations in GPU computing. First, GPU does not allow dynamic memory allocation during the tasks. As a consequence, the necessary memory for a task should be determined in advance. Moreover, memory lock is not efficiently supported. One of the most important limitations is that the communication between CPU and GPU is still too expensive. Therefore, the CPU-GPU communication becomes the bottleneck of the overall computation [11].

Despite the limitations, the many-core parallel processing is useful for problems dealing with large amounts of data. In this section, we present our methods to exploit the parallel processing ability and to overcome the limitations in order to implement an efficient genetic algorithm for large scale travelling salesman problems.

3.1 Parallel computation of fitness values of genes

A single gene describes a tour visiting all the cities. The fitness of the gene is the sum of distances between every adjacent city pairs in the gene sequence. Let us denote the city in the i-th place of the sequence by c_i, and the distance between two cities c_i and c_j by $d(c_i, c_j)$. f_x, the fitness of a gene x, can be computed as follows:

$$f^x = 1/\sum_{i=1}^{n} d(c_i, c_{(i+1)\mathbf{mod}n}) \tag{2}$$

It seems that we can create n threads of which index τ ranges from 1 to n to perform a task parallelly computing $d(c_\tau, c_{(\tau+1)\mathbf{mod}n})$. However, each thread τ accumulates its computation result to the same variable f^x which should be synchronized. The synchronization nullify the efficiency obtained by the parallel processing. In order to solve this problem, we create threads in accordance with the number of genes. Suppose we have m genes, the range of τ is then [1, m]. In the i-th execution of n-sized loop, each thread τ performs $d(c_i, c_{(i+1)\mathrm{mod}n})$ for τ-th gene, and adds the result to f^τ. The fitness values then can be computed in an asynchronous way.

3.2 Efficient competition

The evolution is achieved by selecting better genes to produce offsprings. Each gene competes with each other to survive. In order to implement an efficient genetic method, it is important which two genes are selected and compared to one another. Random selection of competing pairs in GPU implementation is not recommended because GPU does not efficiently support memory lock.

To make genes compete with each other without any synchronization need, each gene has its fixed rival in accordance with the location in the gene pool. After the competition, the location where the winner is also fixed in order not to request synchronization.

Suppose we have m genes in the pool, each gene can be indexed by the integer i in the range of [0, m-1]. For a gene j ranging from 0 to $\lfloor m/2 \rfloor$, gene $j + \lceil m/2 \rceil$ is coupled to be compared, and the winner gene is stored at the location j. This competition process does not require any synchronization and can be efficiently performed in a parallel fashion.

3.3 Efficient parallel crossovers

The crossover methods also cannot be performed by random selection of gene pairs because of the same reason mentioned in the previous subsection describing the implementation of parallel competition. Moreover, the constructive crossover cannot be easily implemented with parallel tasks. The traditional

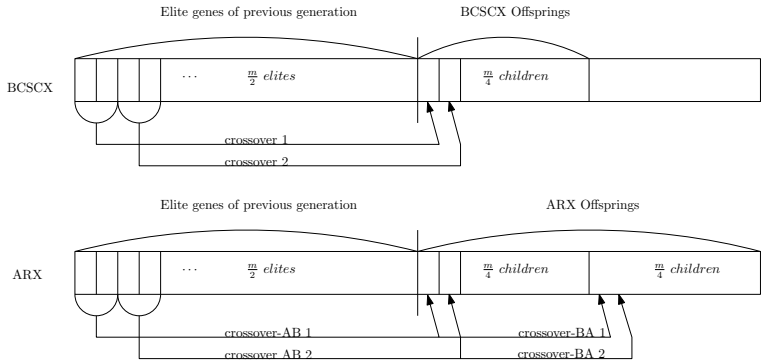


Fig. 3 Parallel crossover without synchronization

Table 2 Thread execution for parallel crossover

n: number of cities, m: number of genes
for each element(k: $1 \cdots n$)
for each thread(τ: $1 \cdots \lfloor m/2 \rfloor$)
do $\xi(k, \tau, \tau + 1, \tau + \lceil m/2 \rceil)$

crossover where the some corresponding subsequences of two parent genes are simply exchanged, and such tasks can be parallelly performed without any difficulties. However, the constructive crossover methods can determin the k-th element in the offspring gene sequence only after the $k-1$-th element has been already determined. Therefore, efficient parallel crossover has two requirements: 1) asynchronous crossover pairing and child reproduction, and 2) parallel implementation of crossover.

In order for the asynchronous coupling of parents, we also fix the partner for each gene. During the competition process, we have $m/2$ elite genes in the first half of the gene pool as shown in the Fig. 3. These genes can For each elite gene who has **even**-number index i ranging from 0 to $\lfloor m/2 \rfloor$, the gene $i+1$ is used as another parent gene. Their offspring is computed and stored at the location $i/2 + \lceil m/2 \rceil$. BCSCX produces only one offspring from two parents. Therefore only $m/4$ offsprings are produced as shown in Fig. 3. However, ARX can produce two offprings from two parents. Therefore one of the offsprings is stored at $i/2 + \lceil m/2 \rceil$, and the other is stored at $i/2 + \lceil 3m/4 \rceil$.

As mentioned, the constructive crossover cannot be parallelly implemented. In order to solve this problem, we create thread in accordance with the number of genes m and each thread independently determines only one element of its offspring gene sequence. In other words, parallel task is denoted by $\xi(k, p_1, p_2, c)$ where ξ is a crossover task, k is the location in the offspring sequence to be determined, p_1 and p_2 are the gene indices of parents, and c is the location for the offspring in the gene pool. Therefore, the parallel crossover can be performed as shown in Table 2.

3.4 Intergroup gene exchange

The effect of parallel crossover becomes significant when the size of gene population is large enough compared to the number of cities. However, simply increasing the population size does not accelerate the convergence while it results in proportional increase of computational burden. The stagnation in fitness improvement even with the increased number of genes is because of the rapid decline in gene diversity. In order to maintain the diversity, we divided the gene pool into several groups and genes are compete with each other and produce offsprings within the specific groups they belong to. To accelerate the convergence, the best gene of each group is periodically transferred to other group. The intergroup gene exchange can be easily implemented with n threads that transfer only one element assigned to them.

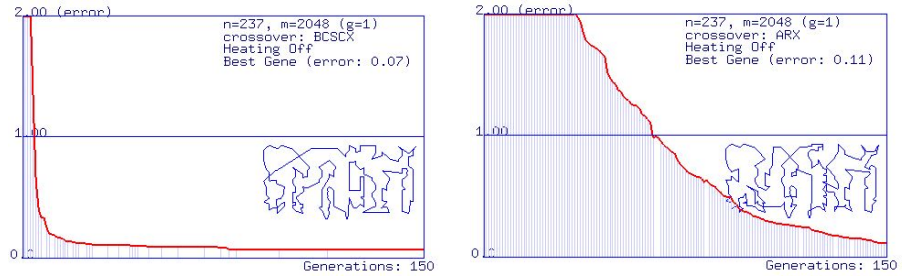
4 Experiments

The experiments were performed on a system running on Ubuntu linux OS with Intel Xeon 3.25 GHz CPU and GTX 980 GPU, and the test data were obtained from TSP data site maintained by University of Waterloo, and they are found at [1].

The convergence trends of BCSCX and ARX were measured as shown in Fig. 4. In this experiment, test data contain 237 cities (xqg237), and 2048 genes were randomly generated. After 150 generations, each crossover method plotted the convergence trends shown in the figure. Although every run of genetic approach show different convergence, they were not very different from those shown in the figure. Therefore, the graphs shown can be regarded typical convergence trends of BCSCX and ARX. The horizontal axis represents the number of generations, and the vertical axis the error ϵ of the found solution with the cost c_b compared to the known optimal solution with the cost of c_o. The error was measured as follows:

$$\epsilon = (c_b - c_o)/c_o \tag{3}$$

As shown in Fig. 4, ARX shows slower convergence than BCSCX because BCSCX greedily selects the best one among the four candidates recommended by two parents. Although, however, the greedy approach rapidly converges in the early stage of the evolution, it does not produce the better solution in the long term. The crossover based on this strategy makes the gene pool homogeneous, and no more effective search will be performed after a local minima is found. This can be easily observed by checking the vertical lines under the graphs. The vertical lines are drawn when the gene pool found a gene which is better than any other genes produced in the previous generations. After the BCSCX rapidly find a gene with small fitness, the record breaking search becomes infrequent. However, the ARX, although slow, continuously updates the fitness record.



(a) BCSCX ($e = 0.07$) (b) ARX ($e = 0.11$)

Fig. 4 Convergence trends (150 generations) of BCSCX and ARX

Table 3 Fitness convergence comparison

	131 cities (xqf131)		662 cities (xql662)	
	gen. to reach ϵ=0.1	ϵ after 100 gen.	gen. to reach ϵ=0.1	ϵ after 100 gen.
BCSCX	35.54	0.041	71.21	0.057
ARX	67.45	0.008	532.42	0.058
hybrid	26.40	0.019	54.23	0.021

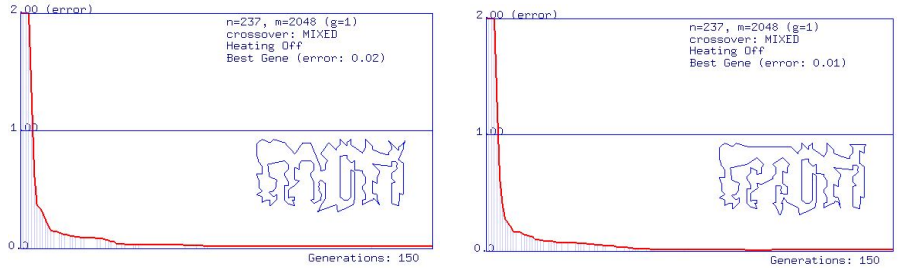


Fig. 5 Convergence trends (150 generations) of hybrid crossover

In order to exploit the advantages of BCSCX and ARX, we implemented a hybrid method that uses both crossover methods. Fig. 5 shows the convergence of the hybrid method with the same date used in Fig. 4. The hybrid method converged more rapidly than BCSCX by including the slower crossover ARX. This shows the significance of the gene diversity in evolutionary methods. The fitness values after 150 generations were 0.021 in average.

Table 4 compares the convergence of crossover methods with 131-city data (xqf131), and 662-city data (xql662). 2048 genes were used. To compare the initial convergence speed, we measured the number of generations required until ϵ is less than 0.1. Each case was test 100 times and the average value was shown in the table. We also measure the error after 1000 generations. Although ARX is slower in the early convergence, the performance can be greatly improved by hybridization with BCSCX.

Table 4 150-generation evolution of 128 local minima genes (662 cities, initial $\epsilon = 0.0915$)

Run	1	2	3	4	5	6	7	8	9	10	avg.
BCSCX	0.047	0.049	0.057	0.056	0.051	0.066	0.038	0.053	0.049	0.053	0.052
ARX	0.025	0.020	0.020	0.022	0.025	0.025	0.025	0.026	0.026	0.022	0.024
hybrid	0.045	0.039	0.048	0.046	0.022	0.044	0.051	0.053	0.042	0.041	0.043

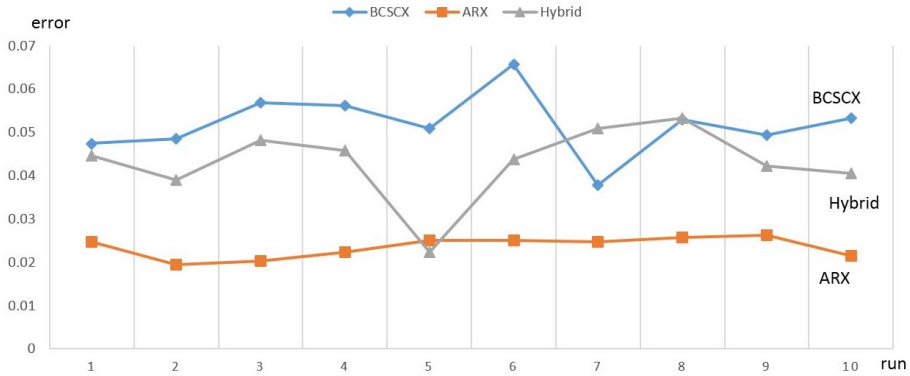


Fig. 6 Fitness after 150-generation evolution of 128 local minima genes(662 cities)

To investigate the advantage of ARX in diversity and continuous search, we applied the ARX to locally optimized genes with preprocessing. We generated 128 local minima genes for 662-city problem (xql662) by applying BCSCX and local improvement such as 2-opt. The local minima genes were then used as initial genes for further evolution with BCSCX, ARX, and hybrid methods, and the convergence were measured as Table 4.

Fig. 6 visually illustrates the convergence of the crossover methods shown in Table 4. The ARX showed the best convergence when applied to differently evolved various genes with low costs.

Similar experiment was done for 10150-city problem (xmc10150). In this case we increase the number of genes by multiplying the 128 local minima genes. As the size of the problem is larger than the 662-city experiment, we measured the ϵ after 500 generations with the different population sizes. The results are shown in Table 5. With this experiment, the increase of population size did not provide satisfactory convergence speed-up. In order to increase the diversity of genes, we divided the 2048 genes into 8 groups, and evolution was applied within each group. The result is shown in the last column in the table. The divided groups improved the performance of the crossover methods.

The performance comparison between GPU implementation and CPU implementation was also performed. We measured the performance for 4 problems which have 131, 2071, 10150, and 100000 cities respectively. Population size was also changed to measure the performance in the various environments. Genes were evolved once as one group, and once as eight groups. The performance was measured with the time consumed to produce the next generation

Table 5 500-generation evolution of local minima genes with different population sizes (10150 cities)

	10150 cities (xmc10150): initial gene group precomputed to be $\epsilon = 0.1841$			
	obtained ϵ after 500 generations			
	128 genes (1 group)	256 genes (1 group)	2048 genes (1 groups)	2048 genes (8 groups)
BCSCX	0.1077	0.1102	0.1107	0.0860
ARX	0.1036	0.1100	0.1019	0.0824
hybrid	0.1038	0.1039	0.1030	0.0839

Table 6 Computation time required for one generation

		number of cities					
		131			2071		
genes	groups	GPU (ms)	CPU (ms)	$\frac{CPU}{GPU}$	GPU (ms)	CPU (ms)	$\frac{CPU}{GPU}$
128	1	2.85	3.22	1.130	26.12	31.29	1.198
	8	4.17	5.32	1.276	29.01	108.71	3.747
256	1	4.03	5.33	1.323	28.32	49.47	1.747
	8	5.08	6.72	1.323	31.08	127.35	4.097
2048	1	15.22	19.15	1.258	63.25	326.35	5.160
	8	15.90	20.33	1.279	66.10	397.50	6.014

		number of cities					
		10150			100000		
genes	groups	GPU (ms)	CPU (ms)	$\frac{CPU}{GPU}$	GPU (ms)	CPU (ms)	$\frac{CPU}{GPU}$
128	1	115.33	208.25	1.806	1160.29	5160.38	4.447
	8	127.85	976.27	7.636	1315.88	29093.50	22.110
256	1	124.78	289.18	2.318	1308.41	6212.37	4.748
	8	132.71	1064.35	8.020	1490.96	30559.15	20.496
2048	1	288.95	1632.42	5.649	5514.09	25905.06	4.698
	8	340.51	2397.90	7.042	5569.41	50387.65	9.047

genes in milliseconds. The column titled CPU/GPU shows the how slow the CPU implementation is compared with GPU version. As shown in the table, the GPU implementation showed better performance when the population size or the number of cities increases.

The proposed method was applied to a large-scale TSP with 100,0000 cities. Fig. 7 shows the result. 2048 genes were used and divided into 8 groups. The best fitness of the initial random gene pool was larger than 17.00. However, only after 200 generations, the gene with $\epsilon = 0.1608$ was found and the solution is shown in Fig. 7. After 5000 generations, the genetic approach produced a gene with $\epsilon = 0.0655$. The currently know best tour is shown in Fig. 7 (d).

5 Conclusion

In this paper, we proposed an effective and efficient genetic approach to travelling salesman problem. The crossover of the proposed method is constructive approach as the previous crossover method for TSP. The constructive methods have serious disadvantages such that they have greedy aspect in the

(a) $\epsilon = 0.1608$ after 200 generations (b) $\epsilon = 0.1359$ after 400 generations

(a) $\epsilon = 0.0655$ after 5000 generations (b) currently known best tour [1]

Fig. 7 The result of the proposed method applied to a large-scale TSP

construction of offspring sequences and it is difficult to parallelly perform the crossovers.

The method proposed in this paper alleviate the greediness of the crossover by alternating the influence of parents in the offspring construction. The method was named ARX, and this method makes it possible for gene pool to maintain the diversity so that the evolution-based search do not stagnate in a local minima. The experimental results show that the hybridization of rapidly converging constructive method and the diversity-conserving ARX can significantly improve the evolution performance.

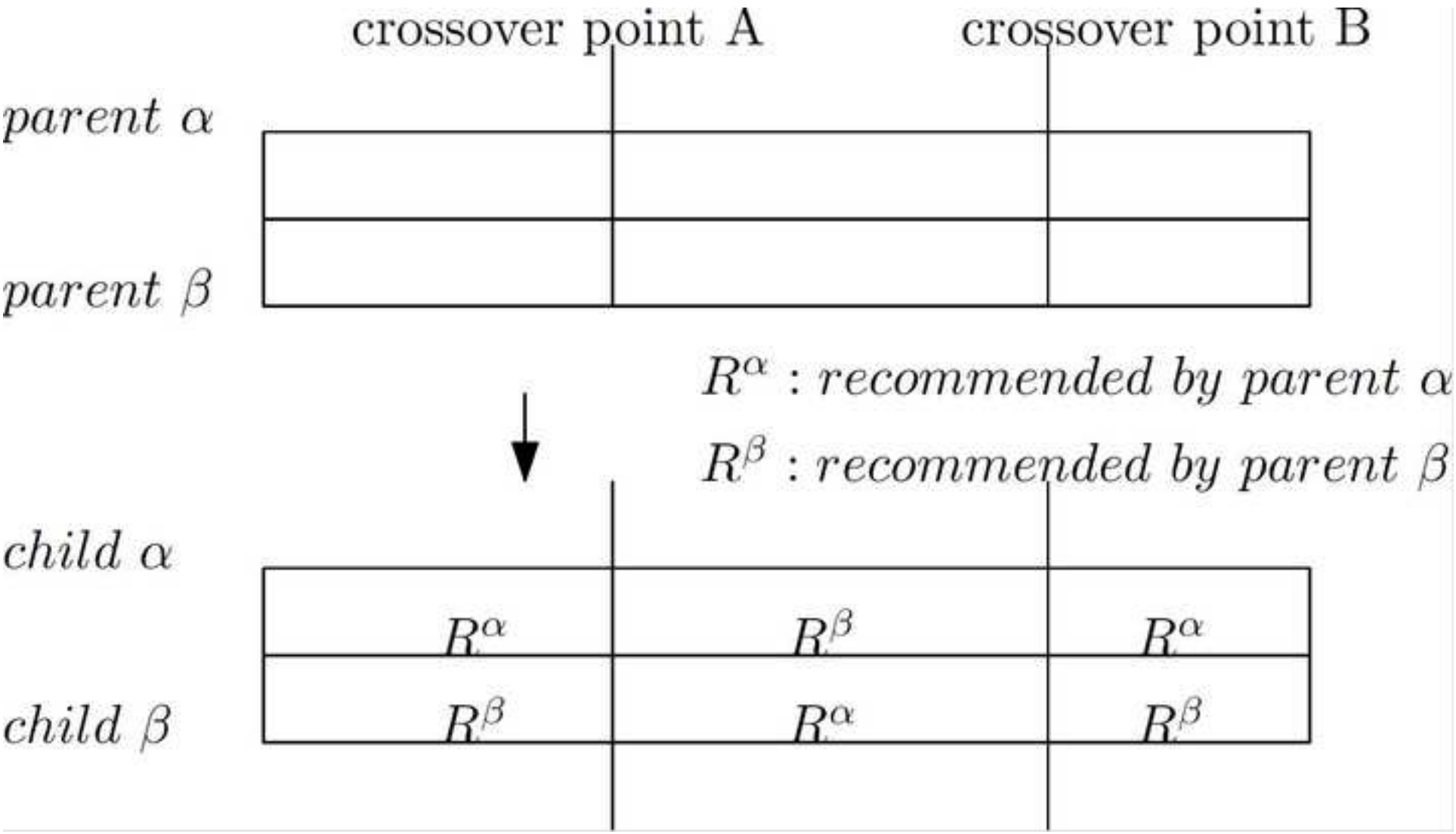
Although the constructive crossover methods cannot be parallely performed, GPU parallelism still can be exploited by creating threads for each gene. In this case, the genes should be sufficiently large enough compared with the problem size (the number of cities). However, simple increase of population size does not guarantee better performance becuase the constructive methods rapidly

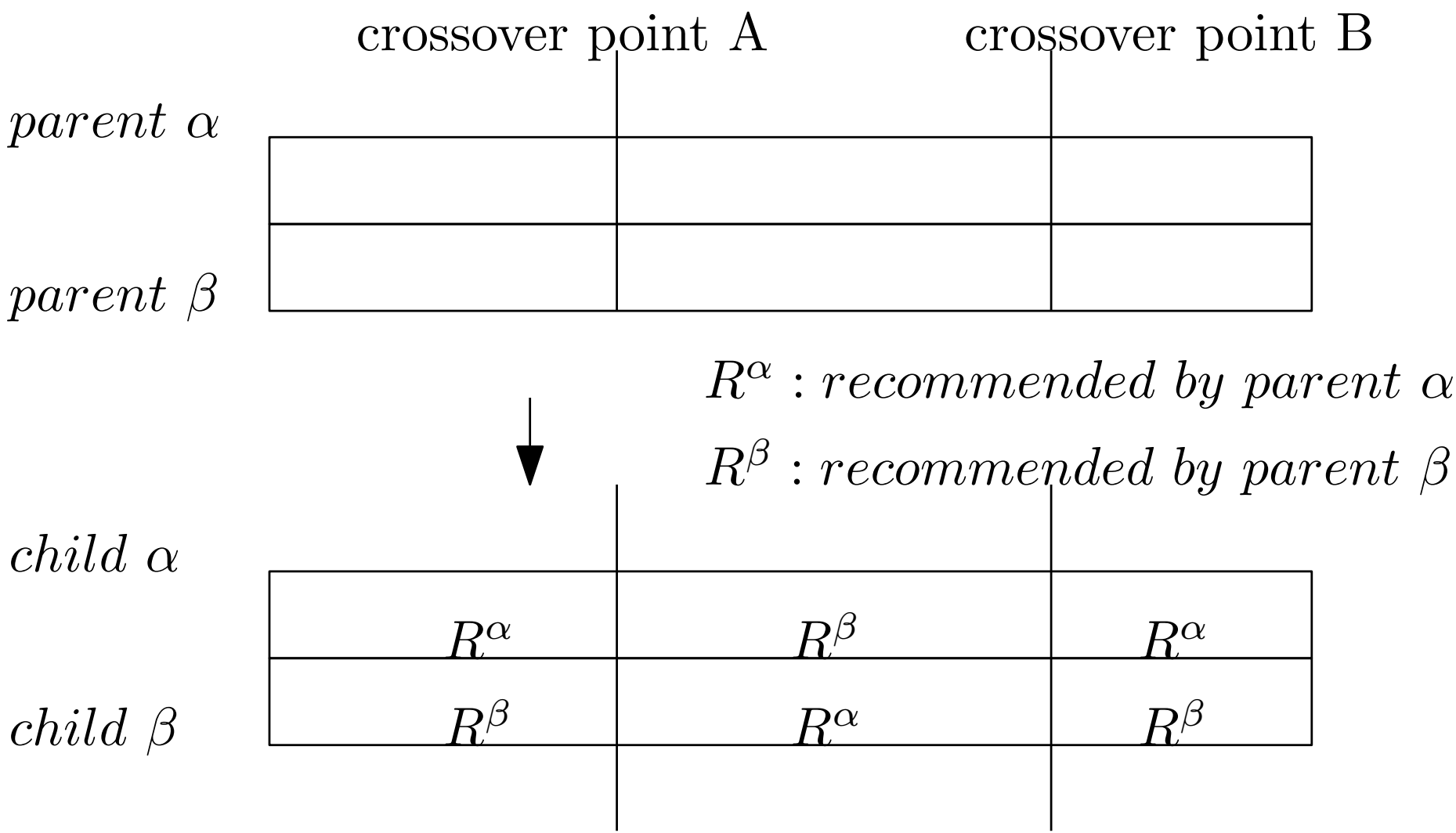
decrease the diversity in the gene pool. We divided the genes into several groups to exploit the effect of increase number of genes, and the experimental results showed that the evolution is improved by separating genes into the groups. The GPU implementation was more efficient in such multi-population environments.

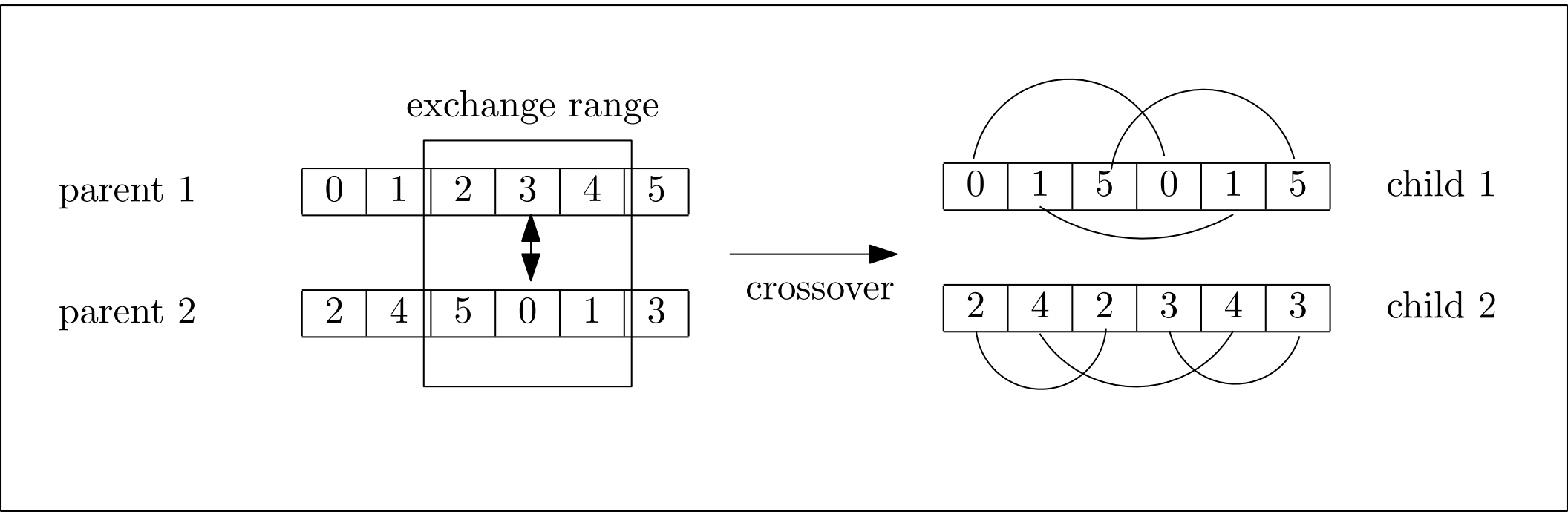
References

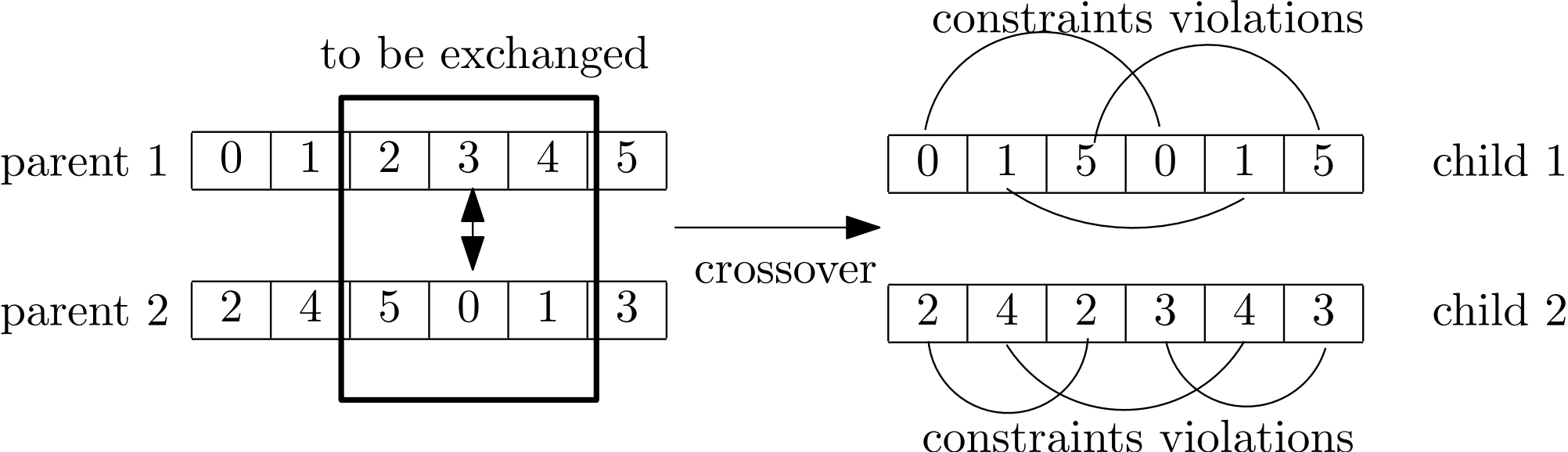
1. Tsp test data. http://www.math.uwaterloo.ca/tsp/data/index.html. Accessed: 2015-11-25
2. Ahmed, Z.H.: Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. International Journal of Biometrics & Bioinformatics (IJBB) **3**(6), 96 (2010)
3. Bäck, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press (1996)
4. Banzhaf, W.: The molecular traveling salesman. Biological Cybernetics **64**(1), 7–14 (1990)
5. Chatterjee, S., Carrera, C., Lynch, L.A.: Genetic algorithms and traveling salesman problems. European journal of operational research **93**(3), 490–510 (1996)
6. Chitty, D.M.: A data parallel approach to genetic programming using programmable graphics hardware. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1566–1573. ACM (2007)
7. De Jong, K.A., Spears, W.M.: An analysis of the interacting roles of population size and crossover in genetic algorithms. In: Parallel problem solving from nature, pp. 38–47. Springer (1991)
8. Fogel, D.B.: Applying evolutionary programming to selected traveling salesman problems. Cybernetics and systems **24**(1), 27–36 (1993)
9. Golberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addion wesley **1989** (1989)
10. Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D.: Genetic algorithms for the traveling salesman problem. In: Proceedings of the first International Conference on Genetic Algorithms and their Applications, pp. 160–168. Lawrence Erlbaum, New Jersey (160-168) (1985)
11. He, B., Fang, W., Luo, Q., Govindaraju, N.K., Wang, T.: Mars: a mapreduce framework on graphics processors. In: Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pp. 260–269. ACM (2008)
12. Hoffman, K.L., Padberg, M., Rinaldi, G.: Traveling salesman problem. In: Encyclopedia of Operations Research and Management Science, pp. 1573–1578. Springer (2013)
13. Kang, S., Kim, S.S., Won, J.H., Kang, Y.M.: Bidirectional constructive crossover for evolutionary approach to travelling salesman problem. In: IT Convergence and Security (ICITCS), 2015 5th International Conference on, pp. 1–4. IEEE (2015)
14. Papadimitriou, C.H.: The euclidean travelling salesman problem is np-complete. Theoretical Computer Science **4**(3), 237–244 (1977)
15. Pereira, F.B., Tavares, J., Machado, P., Costa, E.: Gvr: a new genetic representation for the vehicle routing problem. In: Artificial Intelligence and Cognitive Science, pp. 95–102. Springer (2002)
16. Salomon-Ferrer, R., Gtz, A.W., Poole, D., Le Grand, S., Walker, R.C.: Routine microsecond molecular dynamics simulations with amber on gpus. 2. explicit solvent particle mesh ewald. Journal of Chemical Theory and Computation **9**(9), 3878–3888 (2013)
17. Sanders, J., Kandrot, E.: CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional (2010)

Click here to download Supplementary Material (NOT to be published) ARX.JPG

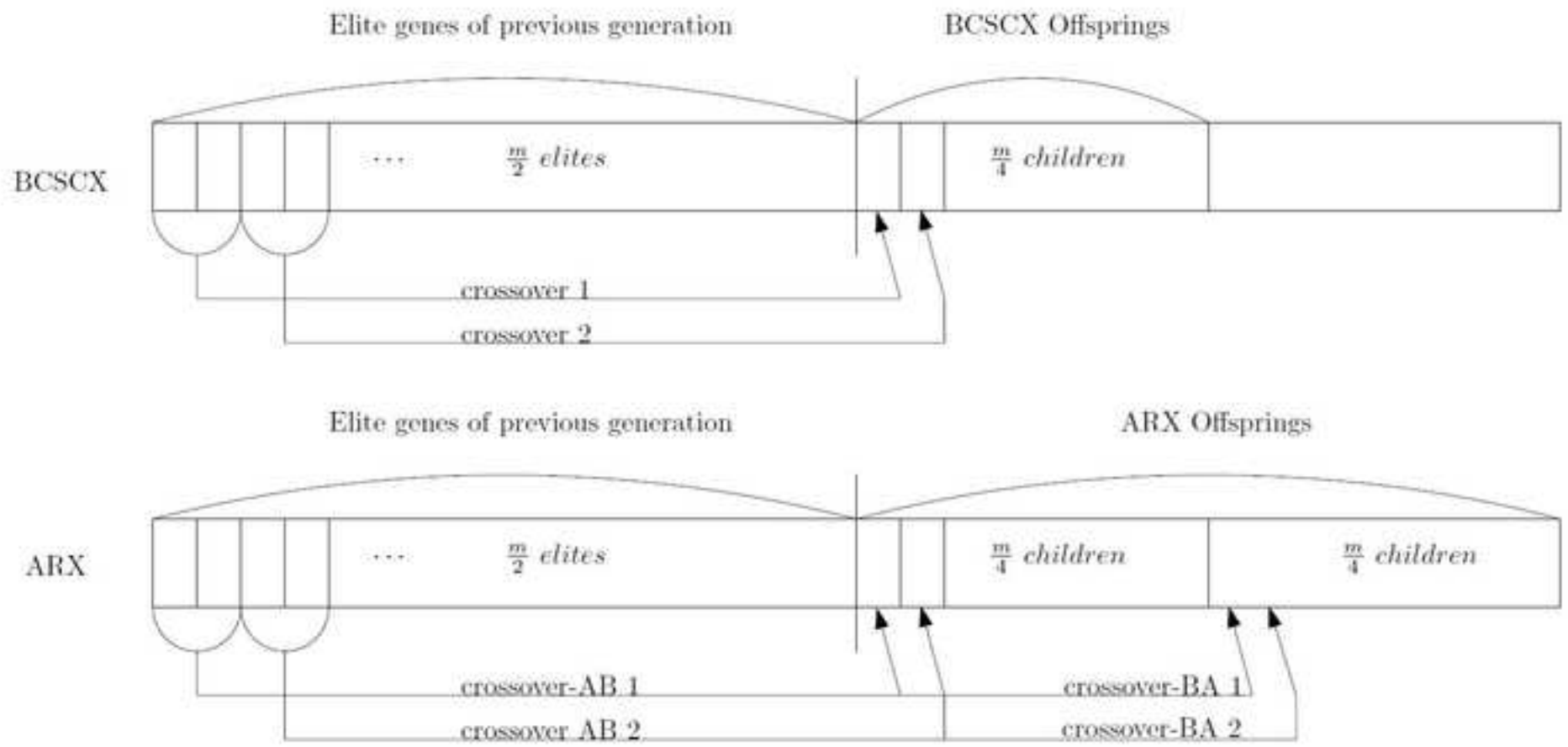


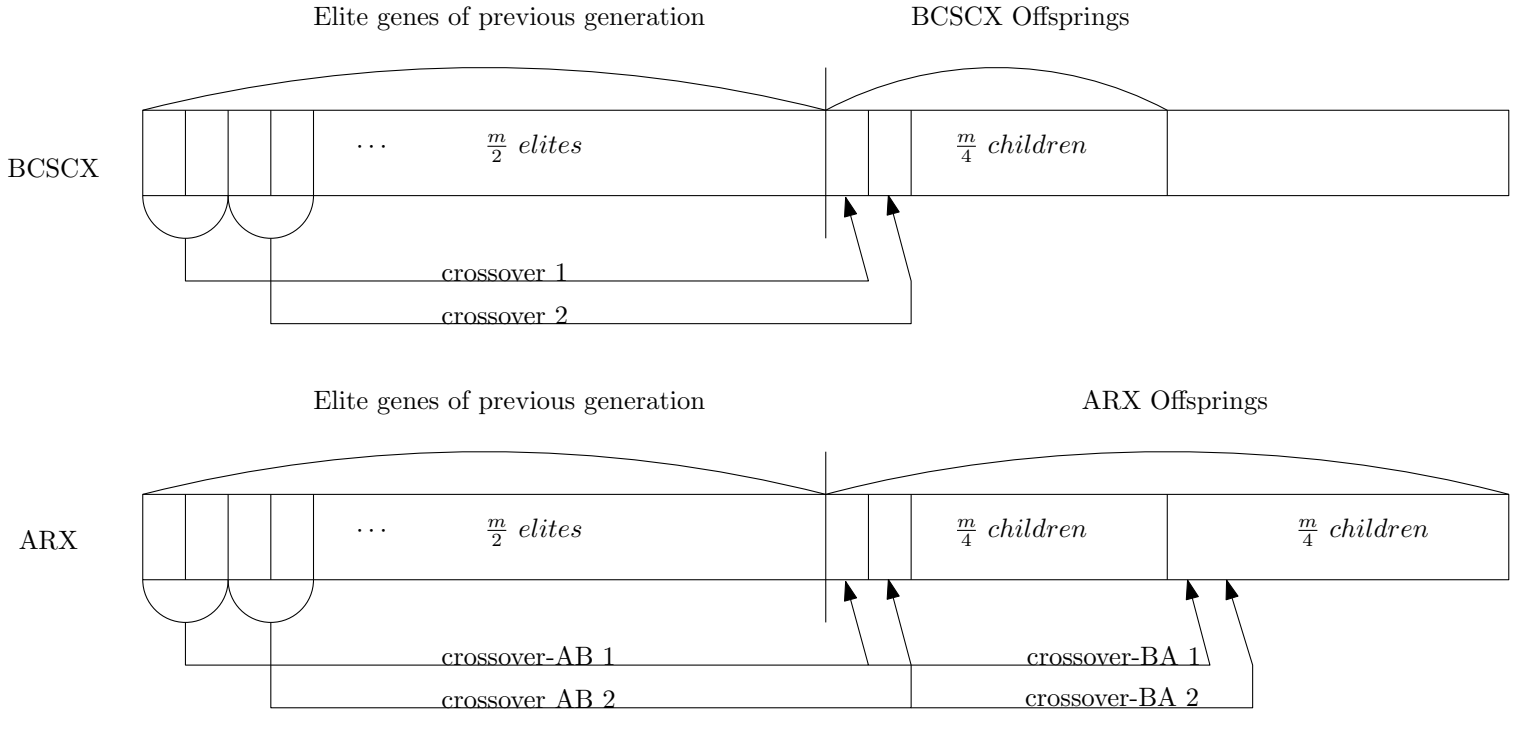

crossover point A
crossover point B
parent α
parent β
R^{α} : recommended by parent α
R^{β} : recommended by parent β
child α
child β
R^{α}
R^{β}
R^{α}
R^{β}
R^{α}
R^{β}


exchange range
parent 1
0 1 2 3 4 5
parent 2
2 4 5 0 1 3
crossover
0 1 5 0 1 5
child 1
2 4 2 3 4 3
child 2


to be exchanged
parent 1
0 1 2 3 4 5
parent 2
2 4 5 0 1 3
crossover
constraints violations
0 1 5 0 1 5
child 1
2 4 2 3 4 3
child 2
constraints violations

Click here to download Supplementary Material (NOT to be published) crossover.jpg




Elite genes of previous generation
BCSCX Offsprings
BCSCX
$\cdots$
$\frac{m}{2}$ elites
$\frac{m}{4}$ children
crossover 1
crossover 2
Elite genes of previous generation
ARX Offsprings
ARX
$\cdots$
$\frac{m}{2}$ elites
$\frac{m}{4}$ children
$\frac{m}{4}$ children
crossover-AB 1
crossover AB 2
crossover-BA 1
crossover-BA 2

Click here to download Supplementary Material (NOT to be published) Monalisa_point047215.jpg