Computers and Operations Research 85 (2017) 87–96

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Solving large batches of traveling salesman problems with parallel and distributed computing

S.G. Ozden[a], A.E. Smith[a,*], K.R. Gue[b]

[a] Department of Industrial & Systems Engineering, Auburn University, Auburn, AL 36849, USA
[b] Department of Industrial Engineering, University of Louisville, Louisville, KY 40292, USA

ARTICLE INFO

Article history:
Received 13 September 2016
Revised 21 February 2017
Accepted 2 April 2017
Available online 6 April 2017

Keywords:
TSP
Parallel computing
Routing
Distributed computing
Facilities design

ABSTRACT

In this paper, we describe and compare serial, parallel, and distributed solver implementations for large batches of Traveling Salesman Problems using the Lin–Kernighan Heuristic (LKH) and the Concorde exact TSP Solver. Parallel and distributed solver implementations are useful when many medium to large size TSP instances must be solved simultaneously. These implementations are found to be straightforward and highly efficient compared to serial implementations. Our results indicate that parallel computing using hyper-threading for solving 150- and 200-city TSPs can increase the overall utilization of computer resources up to 25% compared to single thread computing. The resulting speed-up/physical core ratios are as much as ten times better than a parallel and concurrent version of the LKH heuristic using SPC^3 in the literature. For variable TSP sizes, a longest processing time first heuristic performs better than an equal distribution rule. We illustrate our approach with an application in the design of order picking warehouses.

1. Introduction

With the arrival of multi-core processors in 2005, computers gained more power by providing more clock cycles per CPU. However, most software implementations are still inefficient single processor programs (Ismail et al., 2011). Writing an efficient and scalable parallel program is a complex task. However, C# parallel libraries provide the power of parallel computing with simple changes in the implementation if a certain condition is met: the steps inside the operation must be independent (i.e., they must not write to memory locations or files that are accessed by other steps). Solving large batches of Traveling Salesman Problems is an example of such independent operations. Each TSP instance can be solved by calling a TSP Solver in parallel. Applications of large batches of TSPs include design of order picking warehouses (Ozden et al., 2017), large scale distribution network simulation (Kubota et al., 1999; Sakurai et al., 2006), case-based reasoning for repetitive TSPs (Kraay and Harker, 1997), and delivery route optimization (Sakurai et al., 2011). In these applications the TSP solving consumes most of the computational effort.

We use both the Lin–Kernighan Heuristic (LKH) and the Concorde exact TSP Solver (Concorde). The methods we describe are applicable to optimization problems that must be solved repetitively in an overall algorithm. In this paper, we present two example problems that solve large batches of TSPs and give implementation details in the context of warehouse design for order picking operations. The main result of this paper is to show that doing the parallelism at the TSP level instead of the TSP Solvers' implementation level (Ismail et al., 2011) provides better CPU utilization. A parallel implementation generally achieves better CPU execution times than serial implementations, but an improved CPU utilization is not easily achievable. To the best of our knowledge, this is the first work that presents CPU utilizations for solving large batches of TSPs in serial, parallel, and distributed computing environments.

This work is organized as follows. In Section 2 we give a technical description of the Traveling Salesman Problem (TSP) with solution techniques and its variant of large batches of Traveling Salesman Problems. In Section 3, we describe our implementation of serial, parallel, and distributed large batches of TSPs solvers. In Section 4 we present the computational results, and in Section 5 we offer conclusions.

2. Traveling salesman problem and solution techniques

The Traveling Salesman Problem (TSP) is an NP-hard (Garey and Johnson, 1979) combinatorial optimization problem where a salesman has to visit n cities only once and then return to the

* Corresponding author.
E-mail addresses: gokhan@auburn.edu (S.G. Ozden), smithae@auburn.edu (A.E. Smith), kevin.gue@louisville.edu (K.R. Gue).

http://dx.doi.org/10.1016/j.cor.2017.04.001
0305-0548/

starting city with minimum travel cost (or travel distance). It is one the most famous and widely studied combinatorial problems (Rocki and Suda, 2013). Solving the problem with a brute force approach requires a factorial execution time $O(n!)$ by permuting all the possible tours through n cities and therefore checking $(n-1)!$ possible tours. Given a starting city, there can be $n-1$ choices for the second city, $n-2$ choices for the third city, and so on. In the symmetric TSP, the number of possible solutions is halved because every sequence has the same distance when traveled in reverse order. If n is only 20, there are approximately 10^{18} possible tours. In the asymmetric TSP, costs on an arc might depend on the direction of travel (streets might be one way or traffic might be considered).

Using an integer linear programming formulation (Ismail et al., 2011), the TSP can be defined as:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1}$$

$$\sum_{j \in V} x_{ij} = 1, i \in V \tag{2}$$

$$\sum_{i \in V} x_{ij} = 1, j \in V \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subset V, S \neq \emptyset \tag{4}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \in V \tag{5}$$

where $x_{ij} = 1$ if the path goes from city i to city j and 0 otherwise. V is a set of cities, S is a subset of V, and c_{ij} is the cost of moving from city i to city j. The first set of equalities enforces that each city be arrived at from exactly one city, and the second set enforces that from each city there is a departure to exactly one other city. The third set of constraints ensures that a single tour is created which spans all cities.

TSP is a widely studied problem where solution methods can be classified as Exact Algorithms, TSP Heuristics, or Meta-Heuristics. Exact algorithms are guaranteed to find an optimal solution in a bounded number of steps. Enumeration is only good for solving small instances up to 10 cities. The dynamic programming algorithm of Held and Karp (1962) and the branch-and-bound algorithm are some well known algorithms in this class. They are good for solving instances up to 60 cities. Concorde is a code for solving symmetric TSPs and related network optimization problems exactly using branch-and-bound and problem specific branch-and-cut techniques (Applegate et al., 2007; Cook, 2014). This algorithm is the current exact method of choice for solving large instances. Concorde was able to solve a 85,900-city problem in TSPLIB (2013).

Heuristics are used when the problem is large and a solution is needed in a limited amount of time. We can categorize these heuristics into two groups: “constructive heuristics” and “improvement heuristics.” Constructive heuristics start with an empty tour and repeatedly extend the tour until it is complete. The most popular constructive heuristic is the nearest neighbor algorithm, where the salesman chooses the nearest unvisited city as the next move and finally returns to the first city. Improvement heuristics start with a complete tour and then try to improve it with local moves. The most popular and easily implementable heuristic is the pairwise exchange, or $2-opt$, which iteratively removes two edges and replaces them with two different edges to obtain a shorter tour. The algorithm continues until no more improvement is possible. $k-opt$ is a generalization which forms the basis of one of the most effective heuristics for solving the symmetric TSP, the Lin and Kernighan (1973). $k-opt$ is based on the concept of $k-optimality$: “A tour is said to be $k-optimal$ if it is impossible to obtain a shorter tour by replacing any k of its links by any other set of k links” (Helsgaun, 2000). For a more detailed review of these algorithms refer to Helsgaun (2000).

Meta-heuristic algorithms are designed for solving a problem more quickly than exact algorithms but are not specifically designed for any particular problem class. Most of these meta-heuristics implement some form of stochastic optimization. The solution is dependent on the set of random numbers generated. Meta-heuristics' ability to find their way out of local optima contributes to their current popularity. Specific meta-heuristics used for solving the TSP include simulated annealing (Kirkpatrick, 1984), genetic algorithms (Grefenstette et al., 1985), tabu search (Knox, 1994), ant colony optimization (Dorigo and Gambardella, 1997), iterated local search (Lourenço et al., 2003), particle swarm optimization (Shi et al., 2007), nested partitions (Shi et al., 1999), and neural networks (Angeniol et al., 1988). There are many variants and hybrids of these meta-heuristics designed to solve the TSP (Lazarova and Borovska, 2008).

2.1. Parallel/distributed implementations

The algorithms mentioned in this section solve a single TSP using parallel/distributed techniques. A parallel and concurrent version of the Lin–Kernighan–Helsgaun heuristic using SPC^3 programming is implemented in Ismail et al. (2011). SPC^3 is a newly developed parallel programming model (Serial, Parallel, and Concurrent Core to Core Programming Model) developed for multicore processors. Developers can easily write new parallel code or convert existing code written for a single processor. All of their speed-ups were less than 2 times compared to single thread runs, even when using a 24-core machine. The computational time of each individual task parallelized was insignificantly small, therefore the overhead of the parallelization prevented achievement close to the theoretical boundaries of the speed-up (MSDN, 2016c). In Aziz et al. (2009), a sequential algorithm is developed for solving TSP and converted into a parallel algorithm by integrating it with the Message Passing Interface (MPI) libraries. The authors use a dynamic two dimensional array and store the costs of all possible paths. They decompose the task of filling this 2D array into subroutines to parallelize the algorithm using MPI. The Message Passing Interface provides the subroutines needed to decompose the tasks involved in the TSP solving process into subproblems that can be distributed among the available nodes for processing. Experimental results conducted on a Beowulf cluster show that their speed-ups were less than 3.5 times on a 32 processor cluster. Another technique to implement parallel heuristics for the geometric TSP (symmetric and Euclidean distances between cities), called the divide and conquer strategy, is proposed in Cesari (1996). This reference subdivides the set of cities into smaller sets and computes an optimal subtour for each subset. Each subtour is then combined to obtain the tour for the entire problem. The author was able to achieve between 3.0 and 7.2 times speed-up on a 16 core machine.

2.2. Large batches of traveling salesman problems

Solving a single TSP gives the best path for a certain instance. However, this assumes that the location of the cities (visited points) are fixed. In situations where the problem consists of finding the optimal locations of these cities (visited points), numerous TSPs must be solved to assess a certain design, (e.g, a warehouse layout or a distribution network). Large batches of TSPs are different from the multiple traveling salesman problem (mTSP) which consists of determining a set of routes for m salesmen who all start from and return back to a depot. In large batches of TSPs, to find the expected distance traveled (or another relevant statistic

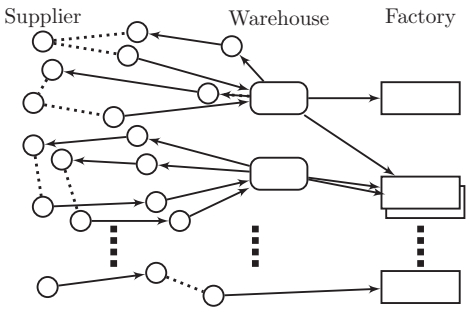


Fig. 1. Large-scale distribution network (Kubota et al., 1999).

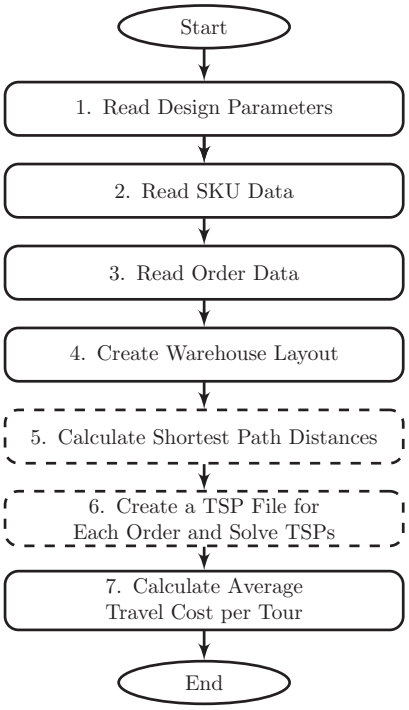


Fig. 2. Stages in warehouse design creation and fitness assessment.

of the distribution of tour lengths), we need to evaluate a large set of tours. Solving large batches of TSPs provides a more accurate estimate and a wider information set than solving only a single TSP. This type of problem can be found in design of order picking warehouses, large scale distribution network simulation (Kubota et al., 1999; Sakurai et al., 2006), case-based reasoning for repetitive TSPs (Kraay and Harker, 1997), and delivery route optimization (Sakurai et al., 2011). Since our focus is solving large batches of TSPs, parallelizing each task at the TSP level will lead to better speed-ups than solving a single TSP using parallel techniques. To best of our knowledge, ours is the first comprehensive comparison of serial, parallel, and distributed large batches of TSPs solvers.

3. Methodology

The methodology for solving large batches of TSPs includes three main steps. In Step 1, problem specific conditions are set (e.g., a warehouse layout structure or a distribution network structure). This typically involves creating locations and calculating distances between locations. We need to assess the total or average cost of TSPs for the given configuration of Step 1. Therefore, multiple TSPs are created and evaluated with a TSP solver in Step 2. In Step 3, the total or average cost of the TSPs is calculated. In the next subsection, we provide an example from the literature in the context of a large-scale distribution network simulation (Kubota et al., 1999). In the later subsections, we discuss this problem in the context of design of order picking warehouses and give implementation details.

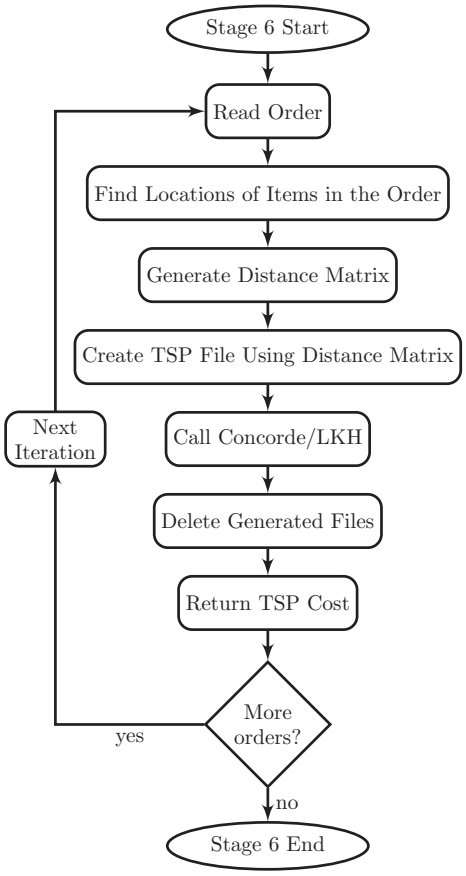


Fig. 3. Serial execution of Concorde/LKH solvers.

3.1. Large-scale distribution network simulation

A large scale distribution network may look like in Fig. 1. This network includes multiple manufacturing enterprises. Parts are delivered from suppliers directly to factories or through warehouses. Direct delivery from the supplier to a warehouse or a factory would be inefficient. Therefore, a truck visits several suppliers and collects parts.

According to Kubota et al. (1999), the total cost of distribution must be calculated. Therefore, several hundreds of distributing routes are created for differing conditions to find the best large-scale distribution network. The authors in Kubota et al. (1999) note that efficiency (i.e., solving overall problem quickly) and precision (i.e., solving close to optimal) are important. The methodology herein is applicable to this class of distribution network problem.

3.2. Design of order picking warehouses

The warehouse design software that motivated this study creates a warehouse layout for given parameters, calculates shortest path distances between storage locations and between storage locations to a pick and deposit location (i.e., a location where a TSP tour starts and ends), creates a TSP file based on the product orders (i.e., each order is a pick tour or a TSP), sends them to the Concorde or LKH solver and reads the resulting TSP distance from these solvers (Ozden et al., 2017). To find the best designs, we must

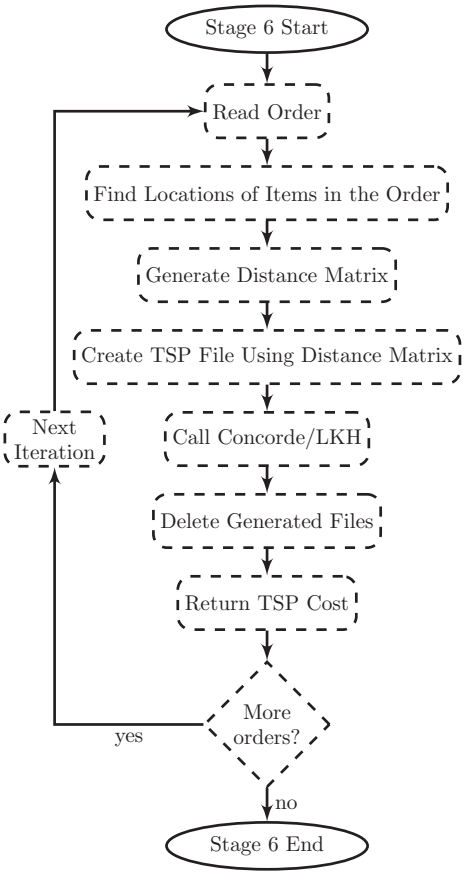


Fig. 4. Parallel execution of Concorde/LKH solvers.

consider numerous product orders that represent the order picking activity of the warehouse. We identify the design that gives the minimum average travel cost.

Fig. 2 shows the seven stages in the warehouse layout creation and fitness assessment. Boxes with dashed lines represent the most time consuming parts of the overall process. In this paper, we focus on Stage 6: "Create a TSP File for Each Order and Solve TSPs." We will describe how we implemented parallel and distributed computing techniques for this stage. Our approach is not specific to warehouse design and can be used for any application that requires solving large batches of TSPs.

We use the C# programming language and the .NET environment. We use C# parallel class methods (MSDN, 2016b) for parallel computing. For distributed computing, we implement a modified version of the asynchronous client-server example from the Microsoft Developer Network (MSDN) (MSDN, 2016a).

3.3. Serial execution of Concorde/LKH

In this case, we send a set of orders (pick lists) to a wrapper function one by one in serial (see Fig. 3). For each order, we find the products and their locations in the warehouse and generate a distance matrix that contains only the items in this particular order. Because the shortest path distances are already calculated in the previous stage, we only generate a sub-matrix of the main distance matrix which contains the all-pairs shortest path distances between every storage location and the pick and deposit location in the warehouse. Based on this distance matrix, we generate a file in the TSP file format (TSPLIB, 2013). Concorde or LKH is called to solve the corresponding TSP file, and to read and keep the distance after the execution. Finally, we delete the generated TSP file and

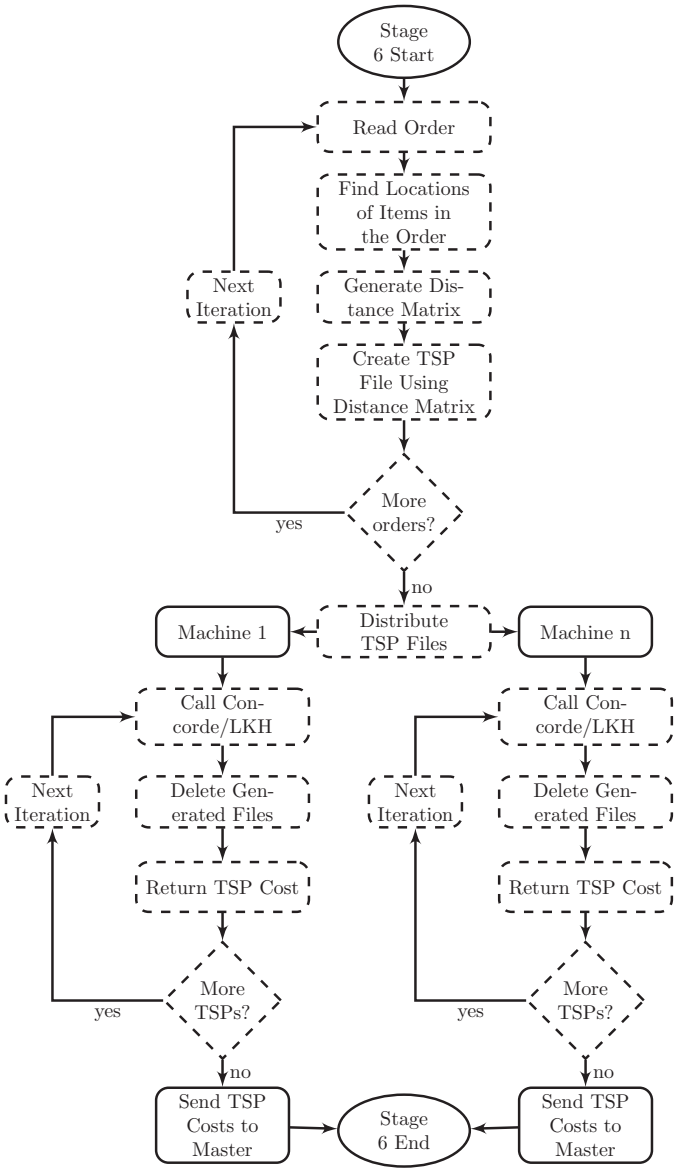


Fig. 5. Parallel and distributed execution of multiple Concorde/LKH solvers.

any generated files from Concorde/LKH and continue to the next order. Stage 6 ends when all orders are evaluated.

3.4. Parallel execution of multiple Concorde/LKH solvers

Since the operation inside the loop of Fig. 3 is independent of any other, we can use "Parallel For Loop"(MSDN, 2016c) and send a set of orders to the wrapper function in parallel. The rest of the operations are the same as a serial execution, but they are performed in parallel until all orders are completed. In Fig. 4, blocks with dashed lines represent the operations that are performed concurrently.

3.5. Parallel and distributed execution of multiple Concorde/LKH solvers

In this case, we have a master-slave architecture to perform distributed computing. We use a static load balancing methodology to distribute TSPs evenly among machines because dynamic load balancing methodologies increase the network overhead by sending and receiving the status of each processor of each slave machine.

Table 1
Problem parameters used for the computational experiments.

Number of cities	5, 25, 50, 100, 150, 200
Number of TSPs solved	10, 100, 1000, 10,000
Execution mode	Serial, parallel, distributed with two or three machines

Also in our first set of experiments, we analyze TSPs of the same size in one batch, which makes dynamic load balancing less effective. We first create TSP files of each order in the master machine in parallel and distribute the TSP files to each slave machine using the TCP/IP protocol with given workload percentage. If the master machine requires provably optimal solutions, then it sends the TSP files using port 8888 otherwise it uses port 8889. Two processes are running on the slave machine. The first listens to port 8888 and solves the TSP files that are sent by the master machine with Concorde in parallel. The second listens to port 8889 and solves the TSP files that are sent by the master machine with the LKH in parallel. A slave machine receives the TSP files over the TCP/IP protocol and keeps the files until Stage 6 ends. The slave machine waits for a "DONE" signal to start TSP runs in parallel, then returns the TSP distance over TCP/IP with the "DONE" signal at the end, closes the communication between the master and slave machines. After the master machine receives all TSP distances, Stage 6 ends. Fig. 5 shows the parallel and distributed execution of multiple Concorde/LKH solvers. Blocks with dashed lines represent the operations that are performed concurrently.

4. Computational results

4.1. Fixed size TSP instances

We have selected fixed sized TSP instances generated using our warehouse optimization software. For the execution of the algorithms, a Lenovo workstation with a six core hyperthreaded Intel Xeon E5-1650 processor is used. Each workstation has 64GB of RAM and 256GB of Solid State Drive. The operating system is

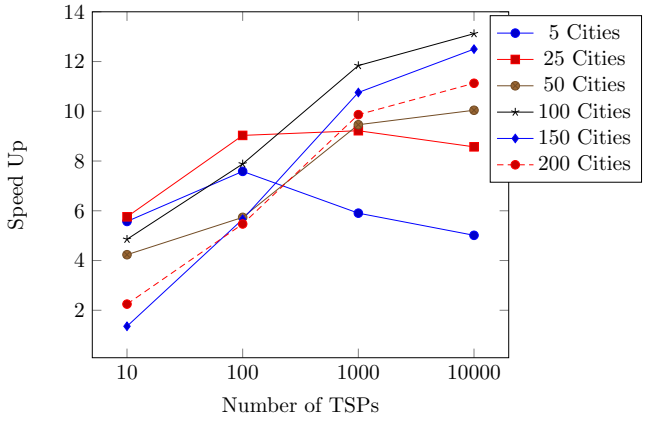


Fig. 6. Number of TSPs vs. number of cities for speed up (Concorde).

64 bit Windows 7, Enterprise Edition. The total number of parallel threads that can be executed is $2 \times 6 = 12$. Table 1 gives a summary of the parameter settings used in the set of experiments. To address the variability in execution times as a result of the background processes of the operating system, we perform five runs for each experiment and calculate average execution time. We use default settings for Concorde. For LKH, we set "RUNS" (the total number of runs) to 1, "TIME_LIMIT" to 1 s and "MOVE_TYPE" to 4 which means that a sequential 4-opt move is to be used.

Table 2 shows the average execution time speed up over serial execution. Speed up is the ratio of the execution time of the serial algorithm to the execution time of the parallel/distributed algorithm. Both LKH and Concorde better utilize computing resources when the TSP size increases (see Figs. 6 and 7). LKH uses parallel executions better than Concorde, because it is set to run with a maximum execution time for each TSP instance. As a result, all concurrent operations have similar execution times, which enables a better workload balance among CPU cores. When solving with Concorde, some TSP instances are harder, therefore the time to

Table 2
Average speed-ups (multiplier).

Cities	Batch size	Parallel		Distributed 2		Distributed 3	
		Concorde	LKH	Concorde	LKH	Concorde	LKH
5	**10**	3.68	3.13	5.87	4.30	7.17	5.80
	100	4.63	3.77	7.82	5.16	10.31	6.96
	1000	3.42	4.25	6.28	6.30	8.01	7.89
	10,000	3.34	4.03	5.31	4.68	6.39	5.48
25	**10**	4.69	3.95	6.28	4.40	6.31	4.87
	100	4.54	4.44	9.87	6.09	12.69	7.40
	1000	3.73	5.63	10.35	7.56	13.58	9.29
	10,000	3.73	5.16	9.22	5.87	12.76	7.70
50	**10**	3.79	4.01	4.78	4.64	4.12	5.35
	100	3.73	6.14	7.09	8.10	6.39	9.73
	1000	4.65	6.75	10.03	9.19	13.70	12.25
	10,000	4.58	6.80	10.52	8.85	15.02	11.57
100	**10**	4.62	4.13	4.70	4.47	5.25	4.66
	100	5.44	6.71	8.06	10.48	10.15	12.95
	1000	6.62	6.98	12.03	11.34	16.87	15.90
	10,000	6.80	7.22	13.21	12.08	19.35	16.74
150	**10**	1.46	4.16	1.25	3.98	1.36	4.34
	100	5.51	6.58	5.87	9.46	5.60	11.81
	1000	6.69	7.07	11.06	10.80	14.51	14.33
	10,000	7.20	7.37	12.53	11.50	17.76	14.79
200	**10**	2.47	3.64	1.45	3.47	2.83	3.59
	100	5.25	6.80	5.20	7.72	5.95	9.37
	1000	6.49	7.13	10.60	9.28	12.51	11.86
	10,000	6.07	7.49	12.30	10.13	15.01	12.92

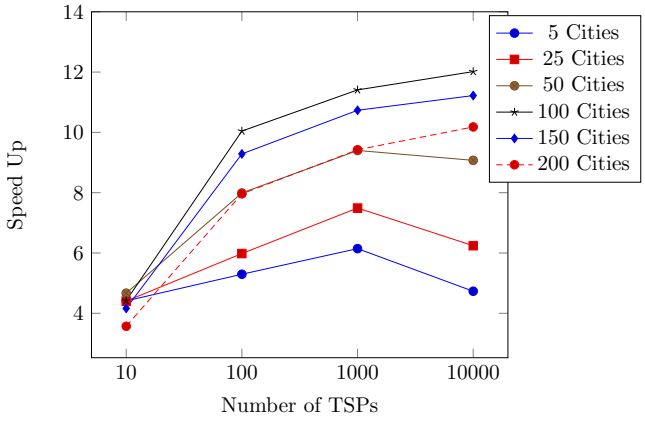


Fig. 7. Number of TSPs vs. number of cities for speed up (LKH).

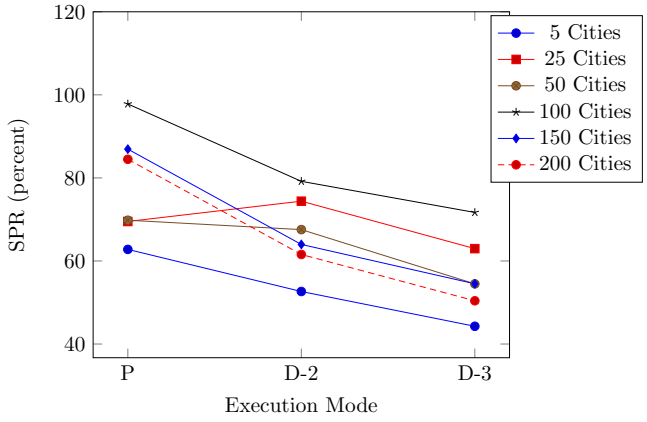


Fig. 8. Execution mode vs. number of cities for SPR (Concorde).

find an optimal solution varies, leading to a poor workload balance among cores (and CPUs for distributed computing).

However, Concorde uses distributed-2 and distributed-3 executions more efficiently because the overhead of sending TSP files to slaves has less effect in execution of Concorde than LKH (see Figs. 8 and 9). In Fig. 6, the speed-up decreases when solving for instances with fewer than 50 cities and more than 100 TSPs. This is because the network overhead of sending many TSP files becomes inefficient for few cities with Concorde. The same is true for LKH, but in this case the speed-up decreases even for a 50 city TSP. There is a point between the number of TSPs (many) and the city size (low) for both TSP Solvers where distributed computing becomes inefficient.

Table 3 shows the average speed-up/physical core ratio (SPR) of the additional CPU cores for parallel, distributed-2, and distributed-3 executions. It is important to note that serial execution uses one of the six physical cores instead of one of the twelve logical cores. A hyperthreaded processor can achieve 30% increased performance compared to a non-hyperthreaded processor (Casey, 2011). Therefore we should be able to see SPR values as high as 130%. Values less then 100% mean that executions with parallel or distributed techniques do not effectively use the physical cores. Values higher than 100% mean that executions with parallel or distributed techniques use computing resources more effectively than serial execution because of hyper-threading. This means that parallelization done at a higher level (solving each entire TSP in parallel) improves CPU utilization over parallelization done at Stage 6 (finding an optimal tour by performing a number of trials where each trial attempts to improve the initial tour) for LKH using SPC[3] (Ismail et al., 2011). Moreover, a parallel implementation can solve more than six times faster than a serial implementation on the same 6-core machine. We used the following formula to calculate the SPR values in Table 3:

$$spr = \frac{su}{npc} \tag{6}$$

where *spr* is the speed-up/physical core ratio, *su* is the average execution time speed-up over serial execution, and *npc* is the number of physical cores available per execution. *npc* values for par-

Table 3
Average speed-up/physical core ratio (percent) and 95% Confidence intervals on SPR.

Cities	Batch size	Parallel		Distributed 2		Distributed 3	
		Concorde	LKH	Concorde	LKH	Concorde	LKH
5	**10**	61(52,70)	52(41,63)	49(41,56)	36(25,47)	40(27,53)	32(22,43)
	100	77(75,79)	63(59,66)	65(62,68)	43(42,44)	57(55,60)	39(37,41)
	1000	57(57,57)	71(70,72)	52(51,54)	52(52,53)	45(43,46)	44(43,44)
	10,000	56(55,56)	67(67,68)	44(44,45)	39(38,40)	36(35,36)	30(30,31)
25	**10**	78(73,83)	66(58,74)	52(46,58)	37(36,38)	35(30,40)	27(21,33)
	100	76(72,79)	74(71,77)	82(80,85)	51(48,54)	70(67,74)	41(40,42)
	1000	62(60,64)	94(89,99)	86(85,87)	63(61,65)	75(72,79)	52(50,53)
	10,000	62(62,63)	86(85,87)	77(76,77)	49(48,50)	71(70,72)	43(41,44)
50	**10**	63(52,74)	67(64,70)	40(36,44)	39(36,41)	23(21,25)	30(28,32)
	100	62(54,71)	102(98,106)	59(55,64)	67(66,69)	35(30,41)	54(51,57)
	1000	77(72,83)	113(111,114)	84(79,88)	77(76,77)	76(67,86)	68(67,69)
	10,000	76(76,77)	113(112,115)	88(85,90)	74(73,74)	83(82,85)	64(63,66)
100	**10**	77(46,108)	69(64,73)	39(33,45)	37(36,39)	29(23,35)	26(26,26)
	100	91(78,103)	112(107,117)	67(55,80)	87(82,92)	56(48,65)	72(66,78)
	1000	110(108,113)	116(114,118)	100(89,111)	95(93,96)	94(90,98)	88(86,90)
	10,000	113(113,114)	120(119,122)	110(109,112)	101(98,103)	107(104,110)	93(92,94)
150	**10**	24(22,27)	69(61,78)	10(5,15)	33(30,37)	8(6,9)	24(24,25)
	100	92(79,104)	110(102,117)	49(39,59)	79(75,83)	31(21,41)	66(61,70)
	1000	112(110,113)	118(116,120)	92(88,97)	90(88,92)	81(74,87)	80(75,84)
	10,000	120(118,122)	123(122,124)	104(99,110)	96(95,97)	99(92,105)	82(81,84)
200	**10**	41(25,57)	61(58,63)	12(6,18)	29(28,30)	16(8,23)	20(19,21)
	100	88(76,100)	113(111,116)	43(22,65)	64(59,69)	33(22,44)	52(46,58)
	1000	108(103,114)	119(116,121)	88(82,94)	77(75,80)	69(46,93)	66(63,68)
	10,000	101(90,113)	125(124,125)	103(96,109)	84(84,85)	83(67,100)	72(71,73)

Table 4
ANOVA: SPR versus size, batch, algorithm, solver.

Source	DF	Adj SS	Adj MS	*F*-value	*P*-value
Size	5	7.6132	1.52264	275.88	0
Batch	3	21.4787	7.15956	1297.22	0
Algorithm	2	12.0637	6.03184	1092.89	0
Solver	1	0.078	0.078	14.13	0
Size*Batch	15	7.7685	0.5179	93.84	0
Size*Algorithm	10	0.7865	0.07865	14.25	0
Size*Solver	5	1.3363	0.26725	48.42	0
Batch*Algorithm	6	0.4232	0.07054	12.78	0
Batch*Solver	3	0.3713	0.12377	22.42	0
Algorithm*Solver	2	1.2914	0.64569	116.99	0
Size*Batch*Algorithm	30	0.7685	0.02562	4.64	0
Size*Batch*Solver	15	1.524	0.1016	18.41	0
Size*Algorithm*Solver	10	0.4784	0.04784	8.67	0
Batch*Algorithm*Solver	6	0.5061	0.08435	15.28	0
Size*Batch*Algorithm*Solver	30	0.4957	0.01652	2.99	0
Error	576	3.179	0.00552		
Total	719	60.1625			
	S	**R-sq**	**R-sq(adj)**	**R-sq(pred)**	
	0.0742909	94.72%	93.40%	91.74%	

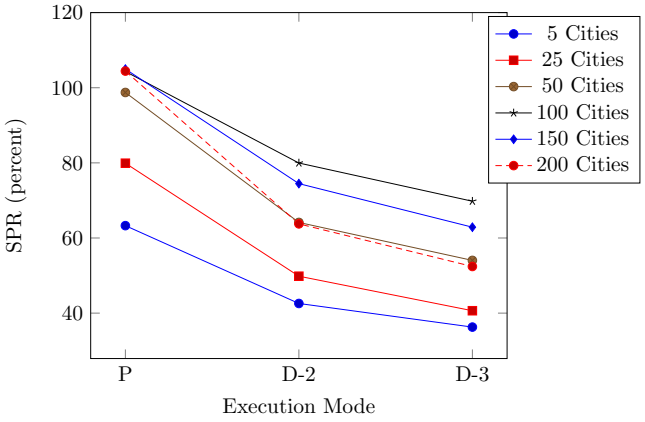


Fig. 9. Execution mode vs. number of cities for SPR (LKH).

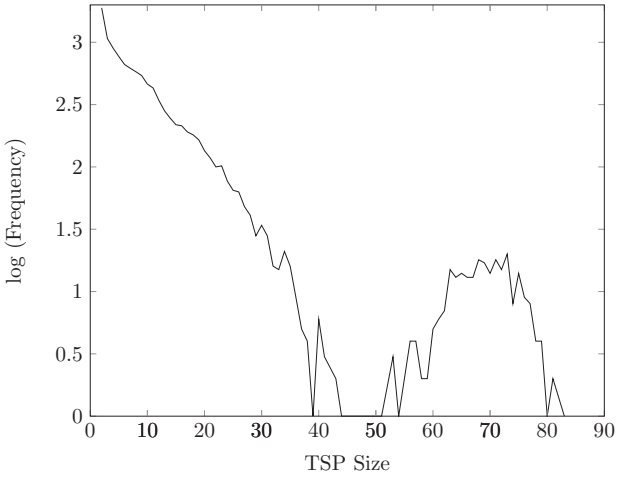


Fig. 10. TSP size frequency for 10,967 TSPs (three outliers, 162, 163, and 164 are omitted from the graph).

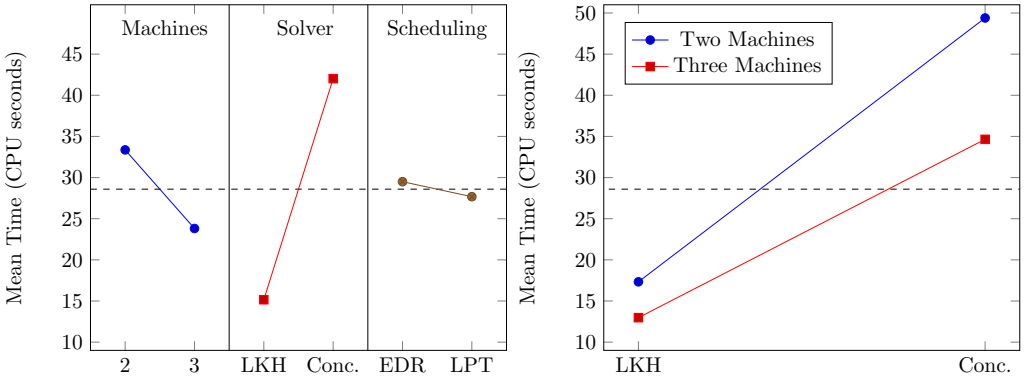
allel, distributed-2, and distributed-3 executions are 6, 12, and 18, respectively. Table 4 shows a 4-way ANOVA using Minitab 17.0 statistical software. The differences between the group means of the main effects (TSP size, number of TSPs (batch), algorithm type (parallel, distributed-2, and distributed-3), solver type (LKH or Concorde)) and their interactions (two, three, and four level interactions) are all statistically significant. The model can explain 94.72% of the variability of the response data around its mean.

4.2. Variable size TSP instances

We should emphasize that our methodology is not bounded to fixed size TSP instances. In this set of experiments, we demonstrate our methodology's ability to solve variable size TSP instances. These instances are from real order picking data where TSP sizes are different. This real order data set has 10,967 TSPs, the average number of cities visited per TSP is 10.12, and the largest TSP has 164 cities. The frequency of TSP sizes is shown in Fig. 10. Because of variable TSP sizes, equal distribution of TSPs among identical machines may create overloaded machines and increase the total makespan. Therefore, we compare an equal distribution rule (EDR) against a well known task assignment rule, the longest processing time (LPT) rule (Graham, 1969). In LPT, jobs (TSPs) are sorted by problem size (as a proxy for processing time, which we do not know yet) and assigned to the machine with the earliest end time so far. In worst case scenario, the algorithm achieves a makespan of $4/3 - 1/3(m)OPT$ where m is the number of machines (i.e., LPT produces a makespan that is at most 33% worse than optimal assignment). We perform five replications for each experiment. Two important things affect the results of these runs: the order of the TSPs in EDR and the estimation of processing time by size of TSP. In some cases, a 40-city TSP can be much harder than a 41-city TSP. Also, a two machine EDR may perform close to optimal, whereas a three machine EDR can be quite suboptimal. The 3-way ANOVA in Table 5 shows that all main factors and the Machines*Solver interaction are significant. Other two-way interactions and the three way interaction are insignificant. The model explains 99.65% of the variability of the response data around its mean. Fig. 11 shows the statistically significant effects — all three main effects and the two way interaction for Machines*Solver. The difference between the means of the TSP solvers is more pronounced than the other two main effects; LKH is nearly three times faster than Concorde. Obviously, increasing from two to three machines reduces the time needed. The 1.8 s difference between the means of EDR and LPT is statistically significant, showing the benefit of using LPT for unequal sizes of TSPs. The two way effect lines are not parallel, showing that Concorde benefits relatively more from using three machines than does LKH.

Table 5
ANOVA: time versus machines, algorithm, solver (real data).

Source	DF	Adj SS	Adj MS	F-Value	*P*-Value
Machines	1	912.11	912.11	996.58	0.000
Solver	1	7219.01	7219.01	7887.54	0.000
Scheduling	1	33.31	33.31	36.40	0.000
Machines*Solver	1	270.03	270.03	295.03	0.000
Machines*Scheduling	1	0.82	0.82	0.90	0.350
Solver*Scheduling	1	2.35	2.35	2.57	0.119
Machines*Solver*Scheduling	1	0.07	0.07	0.08	0.779
Error	32	29.29	0.92		
Total	39	8467.01			
	S	**R-sq**	**R-sq(adj)**	**R-sq(pred)**	
	0.956683	99.65%	99.58%	99.46%	



(a) Main Effects (b) Machines*Solver

Fig. 11. Main effects and Machines*Solver interaction plots for time (real data).

Table 6
ANOVA: time versus machines, algorithm, solver (generated data).

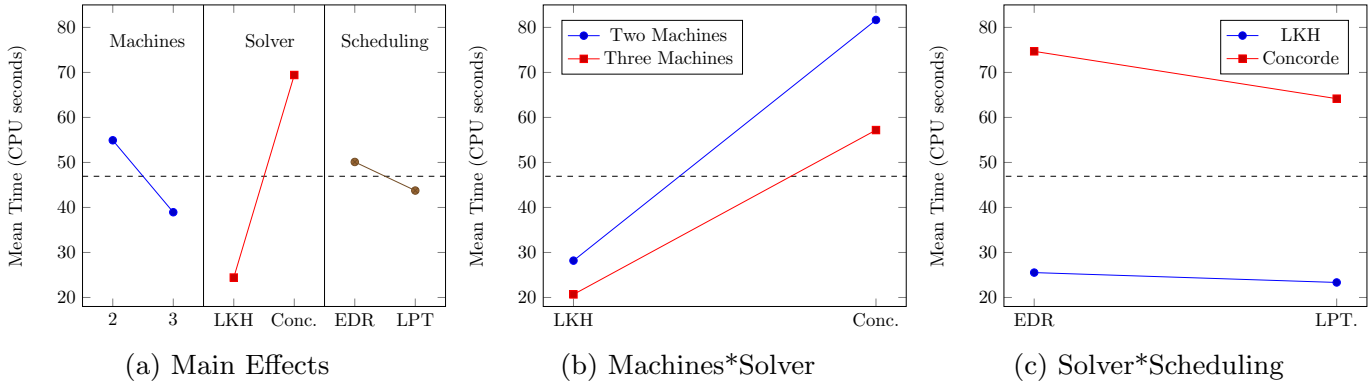
Source	DF	Adj SS	Adj MS	*F*-value	*P*-value
Machines	1	2555.9	2555.9	404.55	0.000
Algorithm	1	20,244.7	20,244.7	3204.36	0.000
Scheduling	1	403.5	403.5	63.87	0.000
Machines*Solver	1	718.9	718.9	113.79	0.000
Machines*Scheduling	1	8.2	8.2	1.29	0.264
Solver*Scheduling	1	173.1	173.1	27.40	0.000
Machines*Solver*Scheduling	1	36.2	36.2	5.73	0.023
Error	32	29.29	0.92		
Total	39	8467.01			
	S	**R-sq**	**R-sq(adj)**	**R-sq(pred)**	
	0.956683	99.65%	99.58%	99.46%	

In order to show LPTs effectiveness, we create a more controlled experiment with generated TSPs. In this case, we create TSPs in the following order repeatedly for 3840 times: size 51, size 11, size 6. In this way, we know that the EDR method will assign all 51-city TSPs to the first machine, all 11-city TSPs to the second machine, and all 6-city TSPs to the third machine. In this experiment, there are 11,520 TSPs and average TSP size is 22.67. All main effects, and two and three way interactions are statistically significant except for the Machines*Scheduling interaction (see Table 6). Fig. 12 shows the three main effects and the significant two way interactions, Machines*Algorithm and Algorithm*Scheduling. The results are congruent with those from the real data. LKH is 2.84 times faster than Concorde on average. LPT finishes on the average 6.35 s earlier than EDR, again showing its benefit. The choice of the scheduling approach is more important when the TSP solver is Concorde because of the longer and non-uniform processing times for this exact method.

5. Conclusions

We presented how parallel computing techniques can significantly decrease the overall computational time and increase the CPU utilization for solving large batches of TSPs using simple and effective parallel class methods in C#. Moreover, we showed our results for distributed and parallel computing methods with two and three slave machines. Using distributed computing techniques requires some background in C# socket programming but simple examples can be found on the web (MSDN, 2016a).

Solving large batches of TSPs is the most computationally intensive step in many applications involving routing. Our results show that using C# parallel class methods is a simple, effective and scalable way to parallelize solving large batches of TSPs. The programmer can write wrapper functions for Concorde and LKH, and implement “parallel for loops” to leverage the multi-core processors. However, distributed computing techniques only show their real



(a) Main Effects (b) Machines*Solver (c) Solver*Scheduling

Fig. 12. Main effects and Machines*Solver interaction plots for time (generated data).

benefits when the TSP instances have more than 50 cities so that the network and file read/write overhead is relatively negligible. Our results also show that for both real data and generated data, a scheduling algorithm like LPT performs better than a naïve method like EDR even though the method used for estimating processing times of TSPs is not very accurate (TSP size, in this case).

The Lin–Kernighan Heuristic (LKH) can be selected over the Concorde TSP Solver when optimality is desired but not required. In our results, LKH is 24.37, 30.59, 20.14, and 21.91 times faster than Concorde on average in single, parallel, 2-computer distributed, and 3-computer distributed runs for solving 10,000 200-city TSPs, respectively. The average optimality gap is less than 0.34% per run.

Acknowledgments

This research was supported in part by the National Science Foundation under Grant CMMI-1200567. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors would like to thank Michael Robbins and Ataman Billor for data collection and testing the code.

Appendix A. Computational time

Tables A.7 and A.8 show the average execution times and average tour costs of Concorde and LKH. In all experiments, we use a single cross aisle traditional warehouse layout. The warehouse can accomodate 1000 items and these items are randomly distributed throughout the storage locations. We generated 10, 100, 1000, and 10,000 orders with 4, 24, 49, 99, 149, and 199 items, and we calculated the distance of each order using TSP solvers. Therefore the average cost for 5 cities and 10 # of TSPs is the average distance of completing 10 orders where each order has 4 items. Since each order has to start and end at the depot, picking 4 items in a warehouse is a 5-city TSP.

Table A7
Concorde average execution times and average cost per tour.

		Average execution time (seconds)				
Cities	Batch size	Single	Parallel	Distributed 2	Distributed 3	Avg. cost
5	**10**	0.27	0.07	0.05	0.04	740.20
	100	2.73	0.59	0.35	0.27	782.50
	1000	27.83	8.13	4.44	3.48	788.10
	10,000	281.20	84.19	52.98	44.00	786.20
25	**10**	0.66	0.14	0.11	0.11	1466.80
	100	6.71	1.48	0.68	0.53	1490.90
	1000	70.33	18.89	6.80	5.19	1499.60
	10,000	684.64	183.48	74.28	53.68	1499.80
50	**10**	1.53	0.41	0.32	0.37	2215.50
	100	16.33	4.47	2.33	2.63	2203.90
	1000	180.18	38.91	17.99	13.35	2208.80
	10000	1763.84	384.82	167.74	117.51	2208.00
100	**10**	6.21	1.44	1.32	1.20	2747.80
	100	150.80	28.02	19.20	14.99	2826.10
	1000	1484.11	224.42	124.68	88.17	2807.20
	10,000	14,081.30	2070.64	1065.72	728.33	2807.40
150	**10**	21.25	14.84	20.66	15.85	3055.32
	100	358.97	66.28	63.61	70.01	3055.05
	1000	3882.53	580.35	351.87	269.67	3053.73
	10,000	39,560.49	5492.05	3164.54	2238.90	3050.34
200	**10**	66.58	30.09	131.03	29.17	3148.05
	100	743.70	143.93	309.28	139.23	3161.00
	1000	7594.09	1173.47	719.45	799.00	3160.18
	10,000	74,592.12	12,502.08	6082.22	5193.40	3157.99

Table A8
LKH average execution times and average cost per tour.

		Average execution time (seconds)				
Cities	Batch size	Single	Parallel	Distributed 2	Distributed 3	Avg. cost
5	**10**	0.11	0.04	0.03	0.02	740.20
	100	1.13	0.30	0.22	0.16	782.50
	1000	11.29	2.66	1.79	1.43	788.10
	10,000	114.15	28.29	24.40	20.84	786.20
25	**10**	0.21	0.05	0.05	0.04	1466.80
	100	1.80	0.41	0.30	0.24	1490.90
	1000	19.00	3.39	2.52	2.05	1499.60
	10,000	185.82	36.03	31.68	24.18	1499.80
50	**10**	0.65	0.16	0.14	0.12	2215.50
	100	5.51	0.90	0.68	0.57	2204.10
	1000	55.85	8.27	6.07	4.56	2209.00
	10,000	551.92	81.21	62.35	47.73	2208.20
100	**10**	2.05	0.50	0.46	0.44	2752.90
	100	25.72	3.84	2.46	2.00	2829.20
	1000	249.86	35.78	22.03	15.73	2809.80
	10,000	2514.98	348.26	208.26	150.25	2809.90
150	**10**	2.90	0.71	0.74	0.67	3064.88
	100	31.64	4.83	2.46	2.69	3065.36
	1000	331.69	46.91	30.73	23.22	3063.17
	10,000	3310.92	449.05	287.87	223.98	3060.00
200	**10**	3.67	1.01	1.06	1.02	3158.04
	100	30.64	4.51	4.00	3.31	3169.81
	1000	291.98	40.98	31.48	24.66	3170.03
	10,000	3060.64	408.72	302.00	236.98	3168.13

References

Angeniol, B., Vaubois, G.D.L.C., Le Texier, J.-Y., 1988. Self-organizing feature maps and the travelling salesman problem. Neural Networks 1 (4), 289–293.

Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J., 2007. The Traveling Salesman Problem: A Computational Study. Princeton University Press.

Aziz, I.A., Haron, N., Mehat, M., Jung, L., Mustapa, A.N., Akhir, E., 2009. Solving traveling salesman problem on cluster compute nodes. WSEAS Trans. Comput. 8 (6), 1020–1029.

Casey, S., 2011. How to determine the effectiveness of hyper-threading technology with an application. https://software.intel.com/en-us/articles/how-to-determine-the-effectiveness-of-hyper-threading-technology-with-an-application/ [accessed 06.13.2016].

Cesari, G., 1996. Divide and conquer strategies for parallel TSP heuristics. Comput. Oper. Res. 23 (7), 681–694.

Cook, W., 2014. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton University Press.

Dorigo, M., Gambardella, L.M., 1997. Ant colonies for the travelling salesman problem. BioSystems 43 (2), 73–81.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to NP-Completeness. WH Freeman New York.

Graham, R.L., 1969. Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. 17 (2), 416–429.

Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D., 1985. Genetic algorithms for the traveling salesman problem. In: Proceedings of the First International Conference on Genetic Algorithms and their Applications. Lawrence Erlbaum, New Jersey (160–168), pp. 160–168.

Held, M., Karp, R.M., 1962. A dynamic programming approach to sequencing problems. J. Soc. Ind. Appl. Math. 10, 196–210.

Helsgaun, K., 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. Eur. J. Oper. Res. 126, 106–130.

Ismail, M.A., Mirza, S.H., Altaf, T., 2011. A parallel and concurrent implementation of Lin–Kernighan heuristic (LKH-2) for solving traveling salesman problem for multi-core processors using SPC^3 programming model. Int. J. Adv. Comput. Sci. Appl. 2, 34–43.

Kirkpatrick, S., 1984. Optimization by simulated annealing: quantitative studies. J. Stat. Phys. 34 (5–6), 975–986.

Knox, J., 1994. Tabu search performance on the symmetric traveling salesman problem. Comput. Oper. Res. 21 (8), 867–876.

Kraay, D.R., Harker, P.T., 1997. Case-based reasoning for repetitive combinatorial optimization problems, part II: numerical results. J. Heuristics 3 (1), 25–42.

Kubota, S., Onoyama, T., Oyanagi, K., Tsuruta, S., 1999. Traveling salesman problem solving method fit for interactive repetitive simulation of large-scale distribution networks. In: Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on, 3. IEEE, pp. 533–538.

Lazarova, M., Borovska, P., 2008. Comparison of parallel metaheuristics for solving the TSP. In: Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing. ACM, p. 17.

Lin, S., Kernighan, B.W., 1973. An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. 21 (2), 498–516.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Handbook of metaheuristics. In: Iterated Local Search. Springer US, Boston, MA, pp. 320–353. doi:10.1007/0-306-48056-5_11.

MSDN, 2016a. Asynchronous client socket example. https://msdn.microsoft.com/en-us/library/bew39x2a [accessed 02.20.2016].

MSDN, 2016b. Parallel class. https://msdn.microsoft.com/en-us/library/system.threading.tasks.parallel(v=vs.110).aspx [accessed 02.20.2016].

MSDN, 2016c. Parallel loops. https://msdn.microsoft.com/en-us/library/ff963552.aspx [accessed 02.20.2016].

Ozden, S. G., Smith, A. E., Gue, K. R., 2017. Non-traditional warehouse design optimization and their effects on order picking operations. Working Paper.

Rocki, K., Suda, R., 2013. High performance GPU accelerated local optimization in TSP. In: Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. IEEE, pp. 1788–1796.

Sakurai, Y., Onoyama, T., Kubota, S., Nakamura, Y., Tsuruta, S., 2006. A multi–world intelligent genetic algorithm to interactively optimize large-scale TSP. In: 2006 IEEE International Conference on Information Reuse & Integration. IEEE, pp. 248–255.

Sakurai, Y., Takada, K., Tsukamoto, N., Onoyama, T., Knauf, R., Tsuruta, S., 2011. A simple optimization method based on backtrack and GA for delivery schedule. In: 2011 IEEE Congress of Evolutionary Computation (CEC). IEEE, pp. 2790–2797.

Shi, L., Ólafsson, S., Sun, N., 1999. New parallel randomized algorithms for the traveling salesman problem. Comput. Oper. Res. 26 (4), 371–394.

Shi, X., Liang, Y., Lee, H., Lu, C., Wang, Q., 2007. Particle swarm optimization-based algorithms for TSP and generalized TSP. Inf. Process. Lett. 103 (5), 169–176. http://dx.doi.org/10.1016/j.ipl.2007.03.010.

TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ [accessed 02.20.2016]. 2013.