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/© 2017 Elsevier Ltd. All rights reserved.

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:

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

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

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

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

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

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 multi-core 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