Computers and Operations Research 129 (2021) 105226

Contents lists available at ScienceDirect

Computers and Operations Research

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

The Traveling Salesman Problem with Job-times (*TSPJ*)

Mohsen Mosayebi *, Manbir Sodhi, Thomas A. Wettergren

Mechanical, Industrial and Systems Engineering, The University of Rhode Island, Kingston, RI, USA

ARTICLE INFO

Article history:
Received 29 April 2020
Revised 1 December 2020
Accepted 8 January 2021
Available online 15 January 2021

Keywords:
Traveling Salesman Problem
Scheduling problem
Assignment problem
Sequence-dependent setup-time
Min Makespan

ABSTRACT

This paper explores a problem related to both the Traveling Salesman Problem and Scheduling Problem where a traveler moves through n locations (nodes), visits all locations and each location exactly once to assign and initiate one of n jobs, and then returns to the first location. After initiation of a job, the traveler moves to the next location immediately and the job continues autonomously. This is representative of many practical scenarios including autonomous robotics, equipment maintenance, highly automated manufacturing, agricultural harvesting, and disaster recovery. The goal is to minimize the time of completion of the last job, i.e. makespan. Although the makespan objective typically is used in scheduling problems, the transportation times can be significant relative to the processing time, and this changes the optimization considerations. We refer to this problem as the Traveling Salesman Problem with Job-times (*TSPJ*) and present a mathematical formulation and local search improvement heuristics for it. Computational experience in solving the *TSPJ* using both commercial solvers and heuristics is reported.

1. Introduction

The Traveling Salesman Problem (*TSP*) and most variants focus on tour-length related objectives. However, there are applications that require a traveler to move through a set of locations, start a job at each location and then return to the origin. The traveler moves to the next location immediately after starting the job and the job continues with no further involvement of the traveler. The maximum completion time in such cases is a consequence of both the travel between the locations and the assignment of specific jobs to locations. This type of problem is important because it models the operation of automated agents that are distributed over a geographically dispersed area, examples of which can be found in autonomous robotics, equipment maintenance, highly automated manufacturing, agricultural harvesting, and disaster recovery. These agents need not be machines, they can be people or groups of people, as long as an 'agent' can execute the job autonomously once the job is initiated. When a number of agents are brought into complete a set of such jobs, there is typically a transporter that delivers the agents to each job location. The agent then executes the job independently while the transporter moves on to deliver the remaining agents to along the route. The mission is accomplished when the last job is completed and the transporter returns to the origin. So, the goal in this scenario is to minimize the time to complete the last job and the time to return to the starting location.

The scenario can be the route of a "mothership unmanned vehicle" that brings other smaller underwater unmanned vehicles (UUV) around to where they need to go to perform jobs (Bays and Wettergren, 2017; Bays and Wettergren, 2015). The operation time is related to the UUV ability and that time varies depending on the specific location at which the job is being performed. The "mothership" does not revisit the area where each UUV is operating so that there is no interference in the operation of the UUV. This problem can be solved with different objective functions based on the operational mission (i.e makespan), and the manager can use this formulation and solutions to determine which tasks may be included in a particular mission if there are operational deadlines.

The scheduling of maintenance technicians over a geographically dispersed area is another potential application scenario, and it has previously been studied with different objectives. Tang et al. (2007) optimized the scheduling of technicians for planned maintenance of geographically distributed equipment as a multiple tour maximum collection problem which has time-dependent rewards using tabu search heuristics. Rashidnejad et al. (2018) integrated maintenance scheduling and vehicle routing problem in order to minimize total cost and maximize availability of assets using genetic algorithms. These examples are related to, yet different from, our traveler problem; however, there is a previously unstudied variant in which a company is responsible for maintaining some type of equipment over a territory and they have to

* Corresponding author.
E-mail address: mosayebi@uri.edu (M. Mosayebi).

https://doi.org/10.1016/j.cor.2021.105226
0305-0548/

schedule the jobs for a given day when all of the technicians are driven in the same van, and they each get dropped off to perform their service. The maintenance time is related to the technician ability and it varies by the service location which is performed at. The real world example for this scenario is the petroleum extraction fields where they stop extraction when either the proactive maintenance or reactive repairs are scheduled. Due to safety concerns, after the process is stopped, the supervisor distributes the technicians to the control, repair, or maintenance locations. They restart the extraction operation after the last maintenance job is completed (makespan). Furthermore, since the deployment of an agent does not introduce any delay, the optimal route will result in each node being visited once. Revisiting any node increases the delay in reaching out to the later nodes and consequently the completion time of their jobs. The assignment of the technicians and the scheduling of the most efficient van route becomes a problem of *TSPJ*.

In natural disasters such as earthquake or tornadoes, Urban Search and Rescue (*USAR*) makes efforts to locate and rescue persons from collapsed buildings or other urban and industrial sites. The goal is to rescue the largest number of people in the shortest time (UNHCR, 2002). The aim of the humanitarian relief logistics in disaster operations management is to acquire and deliver the requested supplies and services, at the places and times they are needed (Gümüş and Çelik, 2017; Wei et al., 2020). Due to the specialized nature of rescue work, teams are multi-disciplinary. Most of the responders have basic training in structural collapse, live wires, and other hazards. However, their proficiency varies based on the specific type of location, such as housing areas, shopping malls, or industrial complexes. Due to the damages to surface infrastructure and the requirement for rapid response the best option is air transportation. However, there are limited resources such as helicopters (Barbarosoğlu et al., 2002; Guo and Peng, 2019); therefore, a site manager may use a single helicopter transfers multiple rescue teams to different locations. This scenario is a *TSPJ* problem where there are teams that must be transported by a transporter (i.e. the helicopter) to the locations and assign the appropriate team to each location based on the location requirements and team capabilities to complete the rescue missions in the shortest possible time. Here too, each node will be visited once because revisiting any node will introduce an unnecessary delay in getting to later nodes, consequently increasing the completion times at those nodes.

The operation of automated manufacturing environments such as machining centers process work-pieces without oversight(Das and Nagendra, 1997) is another application of *TSPJ*. This is representative of many practical scenarios including some related to highly automated manufacturing where the capability and speed of these machines to make different products are different. An operator visits machines capable of unattended operation to check or start a job on the machine and returns to the home station after the given set of jobs has been initiated. The distance between the machines is assumed to be non-trivial. The time of initiating the job on the machine is based on the actual distance between two machines and also reviewing the setting manuals, preparing auxiliary tooling/equipment such as tool loaders, coolant materials, etc. If two sequential machines in the schedule are the same, the initiating time would be short due to learning effect Soleimani et al., 2020, however, if the machines are partially or completely different, the time for preparing the initiation data/tools is longer. In this case, the setup time is related to the sequence of the machines and independent or dependent to the jobs. The jobs are assigned to the machines so that the execution time of each job is dependent on which machine it is performed at. In these scenarios the objective is to finish the jobs at the machines with a minimum final completion time, and both the assignment of jobs to machines and the tour schedule to visit the machines are variables in the optimization. Revisiting any machine after initiation causes a delay for initiation the later machines and consequently the completion time of the jobs at those machines.

Another example comes from the seasonal harvesting of agricultural crops (Basnet et al., 2006). A company assigns the daily workers to the farms over a geographically dispersed area. They are transported and get dropped off at each farm. The distance between the farms are non-trivial. The time of the job at each farm relates to the farm operations and the capability of the workers. The workers are assigned to the farms so that the operation time at each farm is dependent on specific worker (or workers) that is assigned to that particular farm. The goal is to finish harvesting as soon as possible. Clearly, here too there is no advantage to be gained by revisiting a farm twice since doing so will introduce an unnecessary delay. This scenario can thus also be represented by the *TSPJ*.

To illustrate these type of problems, this paper involves one depot, n fixed processing nodes/locations, and n jobs (such that there is exactly one job for each node with the exception of the depot, and each job has only one task). The jobs are assigned to the nodes in such a manner so that the execution time of each job is dependent on which node it is performed at. The goal is to find the sequence of the nodes to visit along with the assignment of the jobs to the nodes so as to minimize the maximum completion time of all jobs and the time to return to the depot.

To demonstrate the special properties of the *TSPJ* , consider an example with one depot, 5 nodes, and 5 jobs. Table 1 shows the travel-time between the nodes, and the job-times at each node. It is assumed that the depot neither requires nor permits any job. Fig. 1 shows the graph of *TSPJ* for this example. The traveler starts from the depot, visits node 1, initiates job 5, continues to node 2, initiates job 3, and so on. In this case, the total travel-time is 36 units and the maximum completion time is 50 units which results from job 2 being processed at node 5.

The remaining parts of this paper are organised as follows: Section 2 reviews the literature for similar problems and explains the unique aspects of the *TSPJ*. A mathematical model for *TSPJ* is developed in Section 3. The formulations presented in this paper have been solved using GAMS (GAMS, 2019) and the *CPLEX* MIP solvers (CPLEX Optimizer, 2019). Section 4 details a local search improvement heuristic and presents results. Section 5 concludes with future research directions and outlines some variations of *TSPJ*.

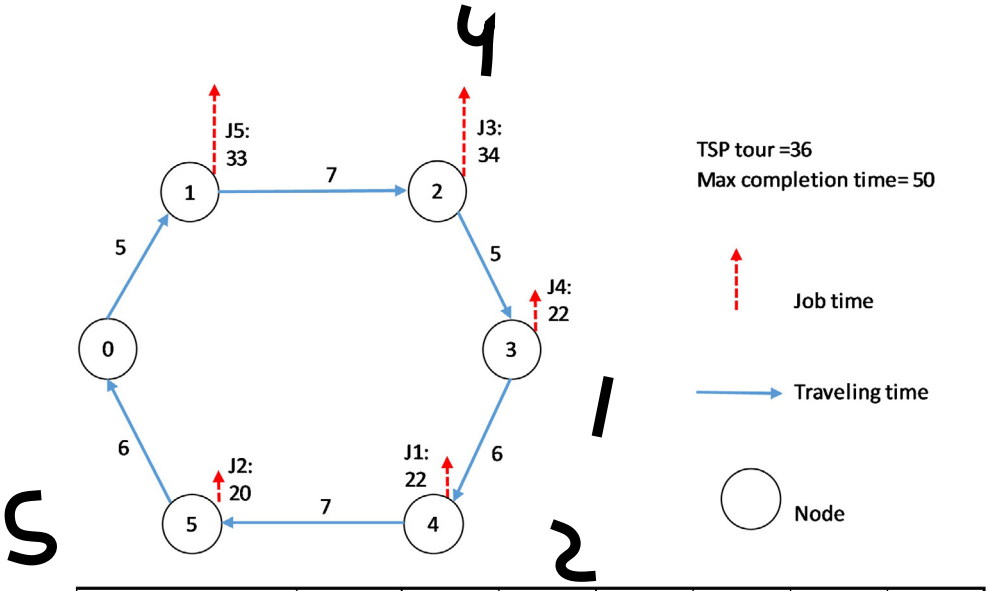
2. Literature review

In the literature, the integrated consideration of scheduling and other problem types has been considered for different applications. Averbakh and Berman (1996) analyzed the routing two-machine flow-shop problems on networks with special structure. They also studied the complexity of the routing open-shop problem on a network (Averbakh et al., 2006). Chernykh and Pyatkin (2019) presented the complexity, solvability and application of the irreducible bin packing to the routing open shop. Shi et al. (2019) optimized a home health care routing and scheduling problem with consideration of uncertain travel and service times.

The *TSPJ* is a hybrid problem that inherits properties from both the *TSP* and the Scheduling Problem. *TSPJ* is a route optimization problem where a traveler must visit all nodes and each node exactly once and then return to the origin. Also, it inherits the scheduling properties when minimizing the completion time of the jobs. When each node executes one job and there is no capacity limitation, the *TSPJ* can be considered as a variation of unrelated parallel machine scheduling problem with setup dependency between the machines that is independent from the job. In this

Table 1
Travel-times between the nodes and job-times at each node.

	Nodes						jobs				
Nodes	0	1	2	3	4	5	1	2	3	4	5
0	–	5	9	12	10	6	–	–	–	–	–
1	5		7	9	12	10	20	22	32	25	33
2	9	7	–	5	10	12	21	20	34	23	32
3	12	9	5	–	6	10	20	22	30	22	34
4	10	12	10	6	–	7	22	24	31	22	32
5	6	10	12	10	7	–	21	20	32	24	34



Nodes by sequence	0	1	2	3	4	5	0
TSP tour to node	0	5	12	17	23	30	36
Job-time	0	33	34	22	22	20	0
Completion time	0	38	46	39	45	50	36

Fig. 1. *TSPJ* illustration.

case, the setup dependent time is similar to the travel time between the nodes in the *TSP*. Thus, *TSPJ* has similarities to the following problems:

- Traveling salesman problem and its variations (such as sequence dependency and time windows);
- Single machine scheduling problem with sequence dependent setup times;
- Unrelated parallel machines scheduling problem with sequence dependency and time windows;

In the discussion that follows, this paper reviews the related literature and highlights the differences from the proposed problem.

2.1. Traveling salesman problem and its variations

The traveling salesman problem and most variants focus on route optimization objectives. In a *TSP*, a traveler sequentially visits a set of spatially distributed nodes. The typical objective of the *TSP* is to minimize the tour length. Dantzig et al. (1954) originally presented the formulation for the large-scale *TSP*. Miller et al. (1960) proposed sub-tours elimination constraints. That initial work on sub-tour elimination was greatly improved by Gavish and Graves (1978). The traveling salesman problem with time windows is a variation of the *TSP* in which nodes must be visited within given time slots (Bretin et al., 2020). These time windows may be 'hard' or 'soft' time windows. Hard time windows are defined by the customers in advance and the time of visiting the node needs to be between the lower and upper bounds of the time window (Taş et al., 2014; Niknamfar and Niaki, 2016). In soft time windows, the traveler can violate time windows, albeit with some penalties (Hu et al., 2018; Liu et al., 2019).

Time dependencies, where the time for visiting a node depends on the visitation times of other nodes, have also been considered for *TSPs* (Hansknecht et al., 2018; Adamo et al., 2020). In this variation of the *TSP*, the cost of any given arc is dependent on its position in the tour. Arigliano et al. (2019) proposed a branch-and-bound algorithm to solve this variation and tested on a set of 4800 instances. Albiach et al. (2008) considered a slightly more general problem in which travel costs may be different from travel times and waiting at a node may incur an additional cost.

The traveling salesman problem with time-dependent service times ($TSP-ST$) is a variation of the *TSP* in which the traveler must stay at the node to provide a service to the customer and the service-time at each customer location is given by a function of the corresponding start time of service. Taş et al. (2016) proposed a lower and upper bound for this problem. Cacchiani et al. (2020) proposed a new mixed integer programming model and branch-and-cut method to solve the $TSP-ST$ problem.

Algorithm 1: *TSPJ − NN*

```
Input: Read Travel Time table(TT) as matrix tt_ij
Input: Read Job-time table (T) as matrix t_ik
Output: Best_sequence
Define C: List of completion time of all jobs in the sequence
C[0] = 0
For S in range(1,Len(TT)) do
  node_pool ← TT
  job_pool ← T
  sequence[0] = S
  For i in range(1,Len(TT)) do
    sequence[i] = argmin(tt_{i-1,j}   ∀j = 1, .., n)
    Remove selected node from node-pool
  End For
  For i in range(Len(TT),1,-1) do
    job[i] = argmin(t_ik   ∀k = 1, .., K)
    Remove selected job from job-pool
  End For
  new-sequence ← sequence + job
  Calculate C_new−sequence
  If C_max > C_new−sequence do
    Best_sequence ← new-sequence
End For
Return Best_sequence
```

A variety of algorithms have been proposed to solve large-scale instantiations of the *TSP* and its variations. Savelsbergh (1985) proposed a local search heuristic for the basic *TSP*. He explored a local search with various side constraints such as time windows on vertices and precedence relations between vertices (Savelsbergh, 1990). Lin and Kernighan (1973) also proposed a local search algorithm. Recently, various local search procedures have been developed to improve the solutions independently (Venkatesh et al., 2019; Jasim and Ali, 2019) or with the combination of other heuristics such as genetic algorithms (Lo et al., 2018), ant colony optimization (Mavrovouniotis et al., 2016), and swarm optimization (Cheng et al., 2016).

2.2. Single machine and unrelated parallel machines scheduling problem

In the unrelated parallel machine scheduling problem a set of n jobs have to be processed in a set m parallel machines. The machines are considered unrelated. The processing times of the job may vary based on the machine to which they are assigned to (Logendran et al., 2007; Mokotoff, 2001). In these problems, the goal is usually to determine the schedule that obtains the minimization of the maximum completion time of the schedule, that is known as makespan or C_{max} (Vallada and Ruiz, 2011).

Many problems related to the scheduling of jobs across multiple machines are computationally intractable (Mokotoff, 2001; Garey and Johnson, 1979), especially when sequence dependencies (Lin and Hsieh, 2014; Lin and Hsieh, 2019), time windows, and uncertain travel times (Wang et al., 2019) are involved. There is also a scenario that the sequence dependency affects the job-time on a single machine (Bianco et al., 1988; Ahonen and de Alvarenga, 2017). In this case, when the job j is executed after job i, the job-time of j is based on the dependency between i: and j since the previous job must be finished before starting the next job. In such problems, the makespan is simply the sum of all of the resulting sequence-dependent job-times.

Sequence dependencies can relate to the relative order of the jobs or the order in which machines are visited. Scheduling problems with job sequence based dependencies are well studied (Kim et al., 2019; Cheng et al., 2004; Pfund et al., 2004), and examples include the sequencing of painting jobs through a paint cell (Spieckermann et al., 2004) among others. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times also is studied in different industries (Bigras et al., 2008; Alkaya and Duman, 2013). The special case of a single job visiting multiple machines with the order in which the machines are visited resulting in sequence dependent processing times is a *TSP*, wherein the sequence dependent times are the travel times between nodes.

Perhaps the most common formulation for unrelated parallel machine scheduling is when the schedule depends on the setup times between jobs. The setup times can be both machine and sequence dependent. That is, the setup time between jobs j and k on machine i may be different than the setup time between jobs j and k on machine i'. Also, the setup time on a given machine i between jobs j and k may be different than the setup time on the same machine between jobs k and j (Rauchecker and Schryen, 2019; Fanjul-Peyro et al., 2019). The *TSPJ* can be described as an unrelated parallel machine scheduling problem in which the travel distance between nodes is given as the sequence-dependent setup time between the machines (and independent from the jobs); this is a variation which has not previously been examined. Thus, when i: and i' are two machines in the sequence and the traveler initiates machine i' after machine i, and additionally there is a machine pair dependent transportation delay between the machine and the depot(assuming depot as a last machine with no job-time), *TSPJ* can be denoted as $Rm|S_{ii'}, t_{i0}|C_{max}$ using the $\alpha|\beta|\gamma$ notation in (Graham et al., 1979).

Although there are many studies on scheduling problems involving sequence-dependent setup times between jobs that are independent or dependent on the machines, the case with sequence dependent setup time between machines that is independent from the jobs is not well studied. In this paper, we assume there is one job for each machine (location), so when each machine receives a single job, the schedule for performing the jobs is also a schedule for visiting the machines. To our knowledge, this aspect of scheduling problems has not been treated in the literature. If the travel time between machines is non-trivial, then the setup time for individual jobs becomes dependent on the sequence of machines visited, and relates the problem to *TSP*. Furthermore, since the number of jobs is equal to the number of non-depot nodes, the allocation of jobs to nodes is related to the assignment problem. If the travel times are trivial, the problem is exactly the conventional assignment problem. However, the combination of the routing and assignment decisions is a specific feature of the problem presented here, and this is the problem that we refer to as the *TSPJ*.

3. Mathematical model

The conventional integer programming formulation for the *TSP* uses a binary indicator X_{ij} to determine whether or not the salesman travels directly from node i: to node j, c_{ij} is the corresponding cost, and Y_{ij} represents the sequence between nodes (Gavish and Graves, 1978). The solution determines the X_{ij}'s, the Y_{ij}'s, and minimizes the total distance traveled.

Gavish and Graves (Gavish and Graves, 1978) used a network flow problem for subtour elimination in *TSP* which are presented in Constraints (4, 5). In these constraints, Y_{ij}'s represent the sequence of visiting the nodes in a complete tour. For fixed values of X_{ij}'s, the positive number of Y_{ij}'s will be integer. The complete

model presented in Constraints (1–6) tries to find variables X_{ij}, Y_{ij} $i,j = 0,1,2,\ldots,n$ that minimize the objective function (1). Eqs. (2, 3) are assignment constraints limiting a single visit to a node. (6) are the type and range definitions of the variables – X_{ij}'s are binary integers, and Y_{ij}'s are positive, real numbers.

$$Min \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}X_{ij} \tag{1}$$

Subject to :

$$\sum_{i=0}^{n} X_{ij} = 1 \quad \forall \ j = 0,..,n \tag{2}$$

$$\sum_{j=0}^{n} X_{ij} = 1 \quad \forall \ i = 0,..,n \tag{3}$$

$$\sum_{j=0}^{n} Y_{ij} - \sum_{j=0}^{n} Y_{ji} = 1 \qquad \forall\, i = 1,..,n \quad i \neq j \tag{4}$$

$$Y_{ij} \leqslant nX_{ij} \qquad \forall\, i = 1,..,n \quad \forall\, j = 0,..,n \quad i \neq j \tag{5}$$

$$X_{ij} \in \{0,1\} \quad Y_{ij} \geqslant 0 \quad \forall ij \tag{6}$$

3.1. A mathematical model for TSPJ

Using the Gavish and Graves formulation, the notions used in the *TSPJ* formulation are:

X_{ij} is a binary decision variable, indicating travel from node i : to node j,

Y_{ij} accounts for the sequence numbers of nodes visited,

Z_{ik} is an indicator variable denoting the assignment of job k to node i,

C_{max} is the maximum completion time of the jobs in all nodes,

TS_i is the actual time of starting the job at node i,

tt_{ij} is the time of traveling from node i : to node j ,

t_{ik} is the processing time of job k at node i.

The formulation for minimizing the completion time in this scenario is:

$$Min\ C_{max} \tag{7}$$

Subject to :

$$C_{max} \geqslant TS_i + \sum_{k=1}^{n} Z_{ik}t_{ik} \qquad \forall\, i = 1,..,n \tag{8}$$

$$C_{max} \geqslant TS_i + X_{i0}tt_{i0} \qquad \forall\, i = 1,..,n \tag{9}$$

$$\sum_{i=1}^{n} Z_{ik} = 1 \qquad \forall\, k = 1,..,n \tag{10}$$

$$\sum_{k=1}^{n} Z_{ik} = 1 \qquad \forall\, i = 1,..,n \tag{11}$$

$$\sum_{j=0}^{n} X_{ij} = 1 \qquad \forall\, i = 0,..,n \quad i \neq j \tag{12}$$

$$\sum_{i=0}^{n} X_{ij} = 1 \qquad \forall\, j = 0,..,n \quad i \neq j \tag{13}$$

$$\sum_{j=0}^{n} Y_{ij} - \sum_{j=0}^{n} Y_{ji} = 1 \qquad \forall\, i = 1,..,n \quad i \neq j \tag{14}$$

$$Y_{ij} \leqslant nX_{ij} \qquad \forall\, i = 1,..,n \quad \forall\, j = 0,..,n \quad i \neq j \tag{15}$$

$$TS_i + tt_{ij} - (1 - X_{ij})\mathbf{M} \leqslant TS_j \quad \forall\, i = 0,..,n\ \forall\, j = 1,..,n \quad i \neq j \tag{16}$$

$$X_{ij}, Z_{ik} \in \{0,1\}, \quad TS_i,\ Y_{ij} \geqslant 0 \quad \forall\, ij \tag{17}$$

In this model the depot node is represented by node 0. The objective function of *TSPJ* is computed by Eqs. (7)–(9). Eqs. (10, 11) are assignment constraints, and force the model to assign only one job to each node and vice versa. Eqs. 12,13 are the standard *TSP* restrictions forcing the traveler to visit all nodes and each node exactly once. Constraints (14–15) are subtour eliminators proposed by Gavish and Graves, 1978. To follow the sequence and the time of starting the jobs in each node, when the traveler moves from node i: to node j, the starting time of the job at node j has to be greater than or equal to the time of starting job at node i: plus the travel-time between nodes i: and j(Constraint 16). $\mathbf{M}$ is a fixed large number, and can be defined as:

$$\mathbf{M} = \sum_{i}\left(\max_{k}\{t_{ik}\} + \max_{j}\{tt_{ij}\}\right).$$

(17) defines the domains of the variables X, Y, Z, and TS.

4. Heuristics for *TSPJ*

The major characteristic of the *TSPJ* is that the traveler initiates a job and departs from the node immediately. The *TSPJ* contains features of both the *TSP* and assignment problems with a makespan objective. If $\max_{i,k}\{t_{ik}\} \leqslant \min_{i,j}\{tt_{ij}\}$, the optimal solution is obtained by solving the *TSP* using the distances alone. If $\min_{i,k}\{t_{ik}\} \geqslant \max_{i,j}\{tt_{ij}\}$ the integer assignment solution with the makespan objective (C_{max}) can be obtained from the job-times directly, and the tour can be derived from it. The *TSPJ* is therefore NP hard (page 56, 236–238 Garey and Johnson, 1979), and efficient heuristics for its solution are required for solving large problems. Four heuristic procedures are proposed. These heuristics use local search improvement heuristics developed for *TSPJ* to improve solutions obtained by construction heuristics similar to the nearest neighbor approaches used for solving *TSP* s. Details of the local search improvement heuristics and the procedures are as follows. An example detailing the steps followed in the execution of one of the heuristics is presented in Appendix A.

4.1. Nearest Neighbor for TSPJ

The nearest neighbor algorithm is widely used for obtaining initial solutions for *TSP* s. The nearest neighbor heuristic (*NNH*) selects any node as a starting point, and then visits the node nearest to it. Subsequently, the next un-visited node closest to the node most recently visited is selected as the next stop. This continues till a tour is obtained. An extension of *NNH*, $NNH - X$, determines a tour by starting from each node, computing the *NNH* tour

associated with that starting node, and then selecting the shortest of all such tours.

Algorithm 2: Reverse assignments

```
Input: Read sequence as list X
Input: Read Job-time table (T) as matrix t_xk
Output: Jobs list as List K
X' = X
n = Len(X)
K = Zeros(n)
While X'
   K[-1] =argmin {t_xk, k ∈ τ(x)}
   For j in range(1,n) do
      t_jk[-1] =∞
   End For
   X'=X'[:-1]
End While
Return K
```

For the $TSPJ$, $NNH-X$ is adapted to consider not just the travel times, but job completion times as well. The $TSPJ-NN$ algorithm (Algorithm 1) also starts from any node and develops a tour by visiting the nearest neighbors until a tour is completed. After the tour is completed, the algorithm starts from the last node along the NNH tour and assigns the job with minimum processing time at the node to it. Subsequently the algorithm moves backward through the NNH tour and assigns the next un-assigned job with minimum processing time to the node till all jobs are assigned (Algorithm 2). The max completion time for this tour is then noted. In the case of $NNH-X$, the process is repeated by starting from each of the nodes. The solution selected is the shortest completion time of all tours generated.

4.2. $2-Opt$ for TSPJ

The $2-Opt$ is a local search algorithm (Savelsbergh, 1985) that improves an initial TSP tour generated by other algorithms such as NNH or $NNH-X$. It does so by evaluating the tour lengths that result from swapping all combinations of node pairs, and accepting any swaps that improve the objective. When the swap results in a tour crossing, route reversals are implemented. For the $TSPJ$, $2-Opt$ is adapted to consider not just the travel times, but job completion times as well. The $TSPJ-2-Opt$ algorithm (Algorithm 3) starts from the first node of a $TSPJ-NN$ tour, and evaluates the maximum tour completion time resulting from swapping node pairs. If an improvement is possible, the swap is accepted, the algorithm starts from the last node along the this tour and assigns the job with minimum processing time at the node to it. Following that the algorithm moves backward through the tour and assigns the next un-assigned job with minimum processing time to the node until all jobs are assigned (Algorithm 2). Pairwise swaps are again evaluated starting with the first node of the tour, and the process is repeated until no nodes are swapped along the tour.

4.3. Local Search Improvement (LSI) heuristic

In the discussion that follows, let:

$\sigma(i)$ be the sequence in which node i : is visited,

$\tau(k)$ be the node at which job k is executed,

i^* be the node that results in the maximum completion time (C_{max}),

k^* be the job assigned for execution at node i^*, and,

C_i be the completion time of the job at node i.

Thus, for node $i^*, TS_{i^*} + t_{i^*k^*} = C_{max}$. The LSI heuristic improves $TSPJ$ solutions by first considering jobs swapping (keeping the node sequences fixed) in Step I, and in Step II, evaluating node swapping for nodes that precede i^*.

Step 0: Initiation

The $TSPJ$ LSI (Algorithm 4) starts with an initial feasible solution which can be obtained using $TSPJ-NN, TSPJ-2-Opt$ or by the combination of $NNH-X+2-Opt$ and subsequently assigning jobs to the nodes using Algorithm 2. The solution is improved by the following two steps:

Step I: Forward Swaps

In this step, the jobs that are assigned to nodes $\{i|\sigma(i) \geqslant \sigma(i^*)\}$ are evaluated for replacement by jobs that are of shorter duration. The following three steps are performed until neither(18) nor (19 and 20) are satisfied:

Algorithm 3: $TSPJ-2-Opt$

```
Input: Read Travel Time table(TT) as matrix tt_ij
Input: Read Job-time table (T) as matrix t_ik
Input: Read sequence
Output: Best_sequence
Define C: List of completion time of all jobs in the sequence
Best_sequence ← sequence
Start_again
While there is improvement do
   For i in range(1,Len(TT)-1) do
      For j in range(i + 1, Len(TT)+1) do
         If swapping is profitable do
            sequence ← Swap nodes
            job_pool ← T
            For i in range(Len(TT),1,-1) do
               Job[i] = argmin(t_ik  ∀k = 1,..,K)
               Remove selected job from job-pool
            End For
            Update jobs in sequence
            Best_sequence ← sequence
            Goto Start_again
         End if
      End For
   End For
End While
Return Best_sequence
```

Step I.1: If $t_{i^*k^*} = min\{t_{i^*k}, \forall k \quad \sigma(\tau(k)) > \sigma(\tau(k^*))\}$, such that any change in job assignment at node i^* cannot reduce C_{max}, the algorithm jumps to Step II.

Step I.2: While $t_{i^*k^*} > min\{t_{i^*k}, \forall k \quad \sigma(\tau(k)) > \sigma(\tau(k^*))\}, C_{max}$ may be reduced by assigning the job that has minimum job-time $(k = argmin_k\{t_{i^*k}, |\sigma(\tau(k)) > \sigma(\tau(k^*))\})$ at node i^* and then assigning the remaining jobs to the rest of nodes in the reverse order of the $TSPJ$ tour. The new job assignments will improve C_{max} if inequalities (18) are satisfied:

$$TS_i + t_{ik} < C_{max} \qquad \forall\, i,k \quad i \neq i^* \quad \tau(k) = i \tag{18}$$

If inequalities (18) are satisfied, the algorithm accepts the job reassignments and returns to Step I.1, otherwise Step I.3 is executed.

Step I.3: While $t_{i^*k^*} > min\{t_{i^*k}, \forall k \quad \sigma(\tau(k)) > \sigma(\tau(k^*))\}$, for each job k, for which $\sigma(\tau(k)) > \sigma(\tau(k^*)), C_{max}$ may be reduced considering a swap of the jobs between the nodes at $\sigma(i^*)$ and other nodes following it in the $TSPJ$ tour. Swapping is profitable if both of the following conditions are satisfied:

$$t_{i^*k} < t_{i^*k*} \qquad \forall\ k \quad \sigma(\tau(k)) > \sigma(\tau(k^*)) \tag{19}$$

$$TS_i + t_{ik^*} < TS_{i^*} + t_{i^*k^*} \qquad \forall\ i \quad \sigma(i) > \sigma(i^*) \tag{20}$$

Condition (19) ensures the recomputed C_{max} (C_{i^*}) will be reduced. Condition (20) checks that the C_i's for the nodes with reassigned jobs do not exceed the C_{max}. The algorithm considers swaps between node i^* and its succeeding nodes until the last node in the sequence. If conditions (19 and 20) are satisfied, the algorithm swaps the jobs and starts checking with the newly assigned i^* and k^* again; otherwise, the algorithm moves to Step II.

Step II: Backward Swaps

C_{max} can be improved by swapping nodes i^* and its predecessors in the sequence of nodes visited. Let v and w be the nodes that immediately precede and succeed node i^*, respectively, and let u be the node that precedes node v, such that $\sigma(u) = \sigma(i^*) - 2,\ \sigma(v) = \sigma(i^*) - 1,$ and, $\sigma(w) = \sigma(i^*) + 1.$ Using Fig. 2 as an illustration, swapping nodes v and i^* reduces C_{max} since by the triangle inequality, $a < d + e$. However, TS_v will increase by $a + b - d$, and $TS_i\ \forall\ i|\sigma(i) \geqslant \sigma(i^*)$ will increase by $\Delta_{i^*,1} = a + c - d - f$. Here, $\Delta_{i^*,1}$ denotes the change in the route resulting from a swap between node i^* and its predecessor node in the sequence. Note that b and e are equal. Let k^* and k represent the jobs assigned to nodes i^* and v, respectively. The algorithm checks the following steps:

Step II.1: Predecessor Node Swaps

The change in the tour length by swapping the sequence of node i^* and its predecessor is computed by Eq. (21). Subsequently, while conditions (22 and 23) and one of the conditions (24–26) are satisfied, the algorithm swaps nodes i^* and v.

$$\Delta_{i^*,1} = a + c - d - f = tt_{ui^*} + tt_{vw} - tt_{uv} - tt_{i^*w} \tag{21}$$

$$\Delta_{i^*,1} + C_i < C_{max} \qquad \forall\ \sigma_i > \sigma(i^*) \tag{22}$$

$$\Delta_{i^*,1} + TS_i + tt_{i0} < C_{max} \qquad ,\sigma(i) = n \tag{23}$$

$$C_v - tt_{uv} + tt_{ui^*} + tt_{vi^*} < C_{max} \tag{24}$$

Note the condition (24) can be simplified to either condition (25) or condition (26), as given by

$$TS_v + t_{vk} - tt_{uv} + tt_{ui^*} + tt_{vi^*} < TS_v + tt_{vi^*} + t_{i^*k^*} \tag{25}$$

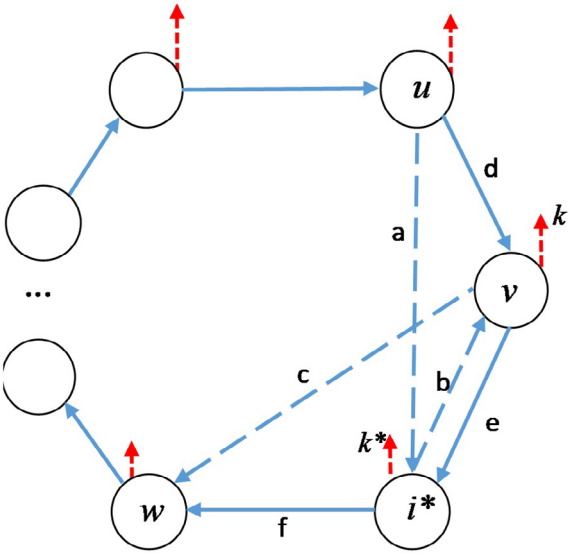


Fig. 2. The backward node swap.

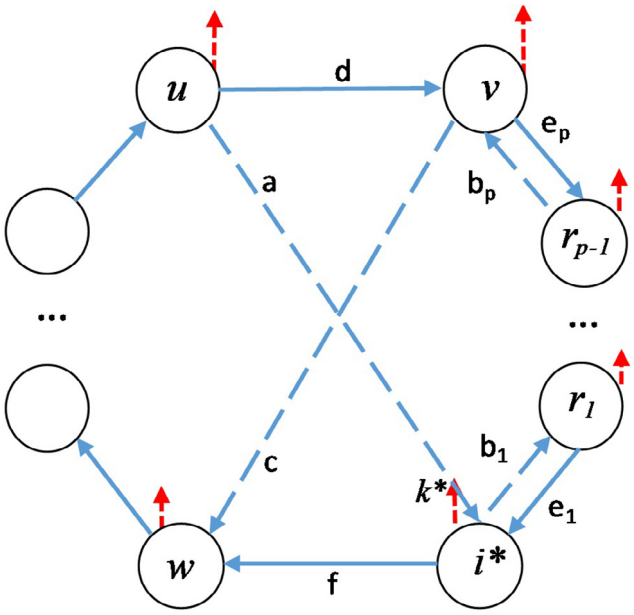


Fig. 3. The generalized backward nodes swap.

$$t_{vk} - tt_{uv} + tt_{ui^*} < t_{i^*k^*} \tag{26}$$

If a predecessor node swap is accepted, the algorithm repeats Step II.1 otherwise, the algorithm moves to Step II.2.

Step II.2: Multi-node Swaps

The concept underlying the predecessor node swap can be extended to consider all exchanges between nodes v and i^* where $\sigma(v) = \sigma(i^*) - p, 1 \leqslant p \leqslant \sigma(i^*) - 1$, as illustrated in Fig. 3.

For each value of p as defined above, let r be any node such that $\sigma(v) \leqslant \sigma(r) < \sigma(i^*)$. Then, Eq. (27) determines the change in the route resulting from a swap of nodes i^* and $\sigma(i^*) - p = v$. Now, if inequalities (28–30) are satisfied, the algorithm swaps all nodes from v to i^* and returns to Step I.1. Clearly, when $p = 1$, this reduces to the case of predecessor node swap (Step II.1).

$$\Delta_{i^*,p} = a + c - d - f = tt_{ui^*} + tt_{vw} - tt_{uv} - tt_{i^*w} \quad \forall\ i = 3,..,n-1 \tag{27}$$

$$\Delta_{i^*,p} + C_i < C_{max} \qquad \forall\ \sigma_i > \sigma(i^*) \tag{28}$$

$$TS_r + t_{rk} < C_{max} \qquad \forall\ \sigma(i^*) - p < \sigma(r) < \sigma(i^*) \tag{29}$$

$$\Delta_{i^*,p} + TS_i + tt_{i0} < C_{max},\ \ \sigma(i) = n \tag{30}$$

When there is no resulting improvement to C_{max}, the final sequence is a local optimum solution.

4.4. Algorithms

This paper proposes and compares four procedures for the *TSPJ* heuristics.

Algorithm 4: Local Search Improvement heuristic

1 Input: Read Travel Time table(TT) as matrix tt_{ij}
2 Input: Read Job-time table (T) as matrix t_{ik}
3 Input: Read sequence
4 Output: Best_sequence
5 Start_again

(*continued on next page*)

Algorithm 4: *(Continued)*

```
While there is improvement do
  If job-time of C_max is not minimum job-time for node do
    Assign minimum job-time to C_max-node
    Reassign other jobs
      calculate C_new−sequence
      If C_max > C_new−sequence do
        Update Best_sequence
        Goto Start_again
    End if
    For S in range (C_max-node + 1,Len(TT)) do
      If swapping jobs conditions are satisfied do
        Swap job between C_max-node and S
        Update Best_sequence
        Goto Start_again
      End if
    End For
    C[0]=0
    For S in range (C_max-node −1,1,-1) do
      If reverse swapping conditions are satisfied do
        Swap C_max-node and S
        Update Best_sequence
        Goto Start_again
      End if
    End For
  End While
  Return Best_sequence
```

Procedure I first constructs a tour for the traveler with consideration of the job assignments to nodes, using $TSPJ-NN$ (Algorithm 1). After all jobs have been assigned, $TSPJ-2-Opt$ (Algorithm 3) is used to improve the objective function.

Procedure II first develops a tour without job considerations. It uses $NNH-X$ to find an initial tour. It then improves the solution by a $2-Opt$ algorithm. Following this tour construction, and starting with last node of the tour, jobs are assigned to the nodes by reverse assignment (Algorithm 2). After this, the solution is improved using local search improvements (Algorithm 4).

Procedure III starts with an initial tour constructed using $TSPJ-NN$ (Algorithm 1). Jobs and nodes are swapped using local search improvements (Algorithm 4). Finally, $TSPJ-2-Opt$ (Algorithm 3) is used to improve the solution.

Procedure IV starts the tour by designating the node that has the greatest distance from the depot as the last node of the tour. It then uses $TSPJ-NN$ (Algorithm 1) to select the remaining sequence of nodes on the tour. After the tour is determined, node and job swapping is performed using local search improvements (Algorithm 4). Finally, $TSPJ-2-Opt$ (Algorithm 3) is used to improve the solution.

4.5. Numerical experience

To test the computational performance of the formulation and the *TSPJ* heuristic procedures, sets of test problems are required. Each test problem consists of a list of nodes and a list of jobs such that the number of nodes and jobs are equal. The inter-node distance is specified. The execution time of each job depends on the node it is executed on, and the table of job times and nodes is specified. A test set contains a number of these problems. Several benchmark problems (test problems) with known optimal solutions are available for the *TSP* in the literature and on widely referenced websites such as *TSPLIB* (Reinelt, 1991). Due to the novelty of the *TSPJ*, a suite of well referenced test problems with both lists of nodes and jobs has yet to be established. Therefore, a set of test problems have been constructed based on *TSPLIB* data for *TSP* (*gr17, gr21, gr24, fri26, bays29, gr48, eli51, berlin52, eli76,* and *eli101*), and adding job-times as a random number between 50 to 80 percent units of the optimal *TSP*-tour which is known for each problem. The reason for selecting job-times as a random number between 50 to 80 percent units of the optimal *TSP*-tour is to introduce competitiveness in the assignment of the jobs to the nodes (refer to the literature review and Section 4). The formulations presented above have all been coded in the GAMS language, and solutions have been obtained using the *CPLEX* MIP solver which was set to terminate when the relative gap of 1% was attained. All runs were performed on a Dell precision tower 7910 with two Intel(R) Xenon(R) CPUs of 2.30 GHz, and 32 GB physical memory (RAM). The aim of using these problem sets is to compare the results of the heuristics procedures with the GAMS/CPLEX solution and quantify the quality of the solutions obtained by the heuristics. The results are listed in Table 2.

Table 2 contains ten columns. The first column from the left is the name of each test problem, which is the same as the *TSPLIB* names with the addition of the label "-J", denoting job times as an additional feature. As an example, *gr17-J* utilizes the *gr17* problem from *TSPLIB*. It contains 17 nodes including depot (16 + 1). The distances between the nodes are the same as original problem and 16 jobs have been added (we assume the depot has no job). For each job, 16 job times are generated – specifying the time for that job if performed at each node of the tour. Each job-time requires between 50 to 80 percent of the optimum tour length, which is known from *TSPLIB*, to be 2760 units. The second column shows the specific heuristic procedure type as explained in Section 4.4. The third column reports the completion time (LSI_C_{max}) that was determined by the heuristic procedure. The processing times of the procedures are presented in the fourth column, and these compare the computational speed performance across procedures (comparisons with other heuristics is a topic of future research). The solutions found by the *MIP* model in GAMS software using the CPLEX solver are presented in the fifth column. The next four columns show the Lower bound, Absolute gap, Relative gap, and the execution time of the GAMS solutions. Finally, the relative gaps between the LSI_C_{max} and GAMS/CPLEX results are shown in the tenth column ($100*(LSI\_C_{max}$ - GAMS$_C_{max})/LSI_C_{max}$).

In Table 3, the results of Table 2 are summarized in a statistical analysis. For each of the procedures, the mean and standard deviation of the gap between the GAMS/CPLEX solutions and the heuristics have been computed across the 10 problems selected and augmented from *TSPLIB*. Also shown is the mean and standard deviation of the *LSI* runtime deviations for each procedure. The *LSI* runtime deviation is defined as the *LSI* runtime for a given procedure operating on a given problem divided by the average *LSI* runtime for that problem (averaged across the four procedures). This yields a relative measure of the computational burden of each of the heuristics when applied to realistic problems. It is clear from these measures that Procedures I, III and IV have the best optimality performance (lowest gap with the GAMS/CPLEX solution), whereas Procedures I and II have the best computational speeds.

Also, to compare the performance of the procedures when applied to different sizes of problems, 300 random problems were generated, and segregated by the number of jobs, into either a small, medium or a large problem category. Each category contains 100 problems. The small, medium, and large problems contain the random number of the nodes between 40 to 50, 400 to 500, and 1000 to 1200 respectively. Also, the distances between the nodes for the small, medium, and large problems range between 10 to 50, 50 to 200, and 200 to 500 units respectively. The *TSP*-tour for each problem is calculated by the local search algorithm using $NNH-X$ and $2-Opt$ Savelsbergh, 1985. The job-times were selected to be between 50 to 80 percent units of the computed

Table 2
Computational results: the heuristics procedures and the GAMS solutions.

Name	*LSI*			GAMS					
	Proc.	C_{max}	Sec	C_{max}	LB	Abs	Rel	Sec	Gap %
	I	2760.0	0.13	2760.0	2760.0	0.0	0.0	0.52	0.0
gr17-J	II	2991.0	0.13	2760.0	2760.0	0.0	0.0	0.52	8.27
	III	2760.0	0.19	2760.0	2760.0	0.0	0.0	0.52	0.0
	IV	2760.0	0.15	2760.0	2760.0	0.0	0.0	0.52	0.0
	I	7956.0	0.21	7788.0	7712.0	75.99	0.0096	0.80	2.16
gr21-J	II	8428.0	0.23	7788.0	7712.0	75.99	0.0096	0.80	8.22
	III	7962.0	0.27	7788.0	7712.0	75.99	0.0096	0.80	2.23
	IV	7962.0	0.21	7788.0	7712.0	75.99	0.0096	0.80	2.23
	I	1818.0	0.36	1806.0	1802.0	3.99	0.0022	0.1.48	0.66
gr24-J	II	1927.0	0.21	1806.0	1802.0	3.99	0.0022	0.1.48	6.7
	III	1818.0	0.47	1806.0	1802.0	3.99	0.0022	0.1.48	0.66
	IV	1853.0	0.34	1806.0	1802.0	3.99	0.0022	0.1.48	2.6
	I	1326.0	0.22	1283.0	1282.94	0.063	0.0004	1.27	3.35
fri26-J	II	1326.0	0.27	1283.0	1282.94	0.063	0.0004	1.27	3.35
	III	1326.0	0.35	1283.0	1282.94	0.063	0.0004	1.27	3.35
	IV	1326.0	0.41	1283.0	1282.94	0.063	0.0004	1.27	3.35
	I	2978.0	0.29	2937.0	2892.88	29.123	0.01	4.44	1.4
bays29-J	II	2943.0	0.35	2937.0	2892.88	29.123	0.01	4.44	0.2
	III	2940.0	0.57	2937.0	2892.88	29.123	0.01	4.44	0.1
	IV	2962.0	0.6	2937.0	2892.88	29.123	0.01	4.44	0.85
	I	7692.0	0.89	7288.0	7215.36	72.64	0.01	13.63	5.54
gr48-J	II	7499.0	2.48	7288.0	7215.36	72.64	0.01	13.63	2.9
	III	7692.0	1.36	7288.0	7215.36	72.64	0.01	13.63	5.54
	IV	7630.0	1.23	7288.0	7215.36	72.64	0.01	13.63	4.69
	I	648.71	3.86	630.0	627.94	2.13	0.0034	13.17	2.97
eli51-J	II	651.0	3.3	630.0	627.94	2.13	0.0034	13.17	3.33
	III	648.71	5.9	630.0	627.94	2.13	0.0034	13.17	2.97
	IV	640.2	5.69	630.0	627.94	2.13	0.0034	13.17	1.62
	I	11230.49	0.96	11087	10976.96	110.54	0.0098	13.72	1.29
berlin52-J	II	11362.0	1.05	11087	10976.96	110.54	0.0098	13.72	2.48
	III	11225.77	1.41	11087	10976.96	110.54	0.0098	13.72	1.25
	IV	11427.66	2.3	11087	10976.96	110.54	0.0098	13.72	3.07
	I	847.99	1.68	802.0	799.47	2.796	0.0035	213.53	5.73
eli76-J	II	845.0	2.61	802.0	799.47	2.796	0.0035	213.53	5.36
	III	847.65	2.05	802.0	799.47	2.796	0.0035	213.53	5.69
	IV	822.46	3.32	802.0	799.47	2.796	0.0035	213.53	2.55
	I	975.94	14.75	947.4	940.59	6.82	0.0072	999.34	3.01
eli101-J	II	1003.0	13.83	947.4	940.59	6.82	0.0072	999.34	5.87
	III	975.94	22.52	947.4	940.59	6.82	0.0072	999.34	3.01
	IV	998.92	28.37	947.4	940.59	6.82	0.0072	999.34	5.44

Table 3
Statistical analysis of results in Table 2.

Procedure	Mean Gap %	Std Dev Gap %	Mean *LSI* Runtime Deviation %	Std Dev *LSI* Runtime Deviation %
I	2.61	1.92	77.0	13.9
II	4.68	2.66	89.9	30.5
III	2.48	2.05	113.2	16.8
IV	2.64	1.64	119.9	25.5

Table 4
Comparing the heuristics procedures.

	Operation time(S)			Average C_{max}			Number of success		
Procedure	S	M	L	S	M	L	S	M	L
I	0.98	105.33	518.38	289.93	3551.98	13908.57	44	19	21
II	0.97	114.88	575.30	292.74	3553.12	13921.98	21	22	13
III	1.57	208.51	1040.25	289.34	3544.75	13892.40	48	34	32
IV	1.54	186.94	957.40	289.39	3544.88	13882.74	45	41	47
Grand Total	1.26	153.91	772.83	290.35	3548.68	13901.42	158	116	113

TSP-tour for each problem. The data for these sample problems is shared in GitHub (https://github.com/*TSPJLIB*). The program to create the data for the problems and the code of the heuristics were written in Python 3.7 (Van Rossum and Drake, 1995).

Table 4 shows the results of the procedures applied to the 300 randomly generated problems. In these results, for each of the four heuristics procedures, the table shows three categories of data including operation time, average C_{max}, and the number of suc-

Table 5
The result of the heuristic IV in comparison with the Nearest Neighbor algorithm and assignment method in large sets.

	Upper bound		Heuristics		
Sample	*TSP*	*TSP+AP*	*TSP*-tour	Procedure IV	Improvement%
pr1002	259045	447975.28	277748.83	407330.83	9.07
pr2392	378061.82	679829.18	411765.46	596003.66	12.33

cesses (success being a run that achieves the lowest C_{max}) for each procedure in each categories of problem sizes. It is clear from these measures that Procedures III and IV have the best optimality performance (lowest average C_{max} specifically in medium and large problems), whereas Procedures I and II have the best computational speeds. For the small problem sets, Procedures III and IV have a higher rate of success. However, for the medium and large problems, Procedure IV has the highest number of successes. Thus, based on the results of Tables 3 and 4, when considering computational time, accuracy on benchmark problems, and performance on randomly generated problems, one can conclude that if one procedure is to be implemented, the recommendation would be Procedure IV. However, each of the heuristic procedures has its own performance benefits and limitations.

There is no benchmark for a very large problem (over 1000 nodes) of this type in the literature. Also, the commercial software are not capable to solve the very large problem in acceptable time. However, this paper constructs the tour with typical nearest neighbor algorithm and then adds the jobs to the nodes using the Assignment Problem (*AP*) solved with the Hungarian method independently. The result can be used as an upper bound for very large scale problem. Table 5 has two parts. The first three columns on the left show the *TSPLIB* problems used as a very large scale sample, the corresponding optimum tour length found in the literature, and the completion time using *AP*. On the right side, the tour found by the *LSI* algorithm IV, the completion time, and the improvement achieved by local search algorithm are presented. Based on these results, it can be seen that the improvement of the heuristics over nearest neighbor algorithm and Hungarian method is more than 9% for two very large problems (pr1002 and pr2392 from *TSPLIB*).

5. Conclusion

This paper introduces a novel variant of the Traveling Salesman Problem that also involves scheduling jobs at the locations. This problem is referred to as the *TSPJ*. A mathematical model of the basic *TSPJ* problem is detailed, and this has been solved using a commercial mixed-integer solver. Examples of applications of the *TSPJ* have been given to frame the problem in a practical context. Four local search heuristics are compared. The gap between the optimum solution of the *CPLEX* solver and the proposed heuristics is less than 5.5% in the sample sets. Future work includes further development of efficient solution procedures and evolutionary algorithms for the *TSPJ* variations such as multiple travelers, multiple depot, and time windows. Also, The job-times for many practical applications are stochastic, those cases can be considered as a variation of *TSPJ* with stochastic job-times which is another problem of future work.

Disclosure statement

The authors do not have any conflict of interest to report for this work.

Funding

This project is not funded by any external source.

CRediT authorship contribution statement

Mohsen Mosayebi: Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Manbir Sodhi:** Conceptualization, Resources, Writing - review & editing, Supervision, Project administration. **Thomas A. Wettergren:** Conceptualization, Formal analysis, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

None.

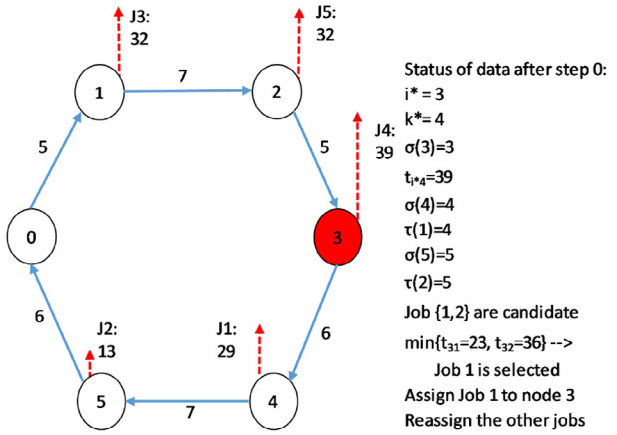
Appendix A. A heuristic solution example

This example uses Table A.6 to show all possible actions in the *LSI* heuristic for *TSPJ*.

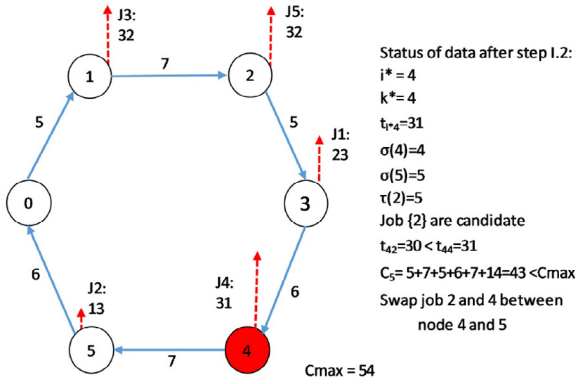
Fig. 4-a shows the initial solution, at Step 0, after the solution using $NNH + 2-Opt$ has been obtained, and jobs have been assigned using Algorithm 2. C_{max} at this point is 56 units, which results from job 4 being assigned to node 3. i^* is node 3 and $\sigma(i^*) = 3$. k^* is job 4 and $t_{34} = 39$ units. Node 4 and node 5 succeed node 3 and jobs 1 and 2 are assigned to them respectively. The exchange condition, $t_{34} > min\{t_{3k}, \forall k = \{1,2\} \quad \sigma(\tau(1)) > \sigma(\tau(3))$ and $\sigma(\tau(2)) > \sigma(\tau(3))\}$ and inequality (18) are satisfied, C_{max} can be reduced by assigning Job 1 to node 3, and determining the assignments for the remaining nodes using Algorithm 2.

Table A.6
Example: Travel-times between the nodes and job-times at each node.

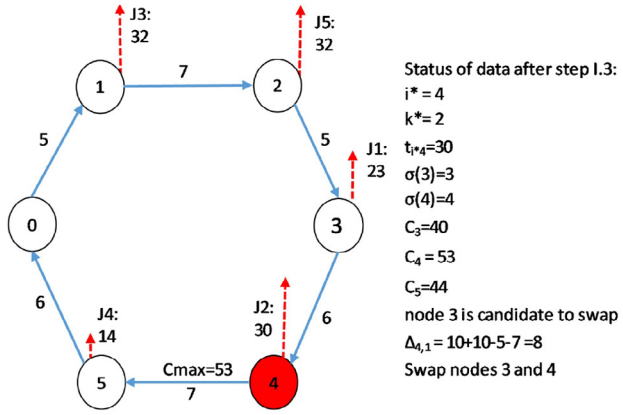
	Nodes						jobs				
Nodes	0	1	2	3	4	5	1	2	3	4	5
0	–	5	9	12	10	6	–	–	–	–	–
1	5		7	9	12	10	20	22	32	25	33
2	9	7	–	5	10	12	21	20	34	23	32
3	12	9	5	–	6	10	23	36	40	39	41
4	10	12	10	6	–	7	29	30	32	31	32
5	6	10	12	10	7	–	22	13	32	14	34



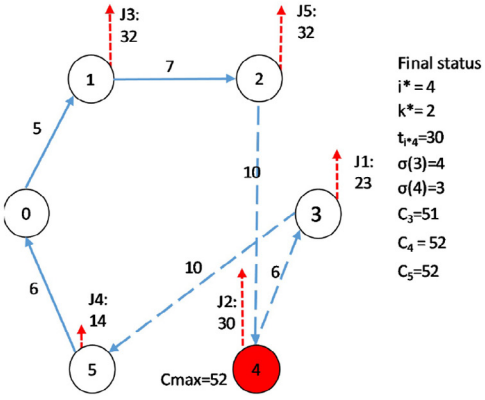
(a) Step 0:NN+2Opt+Job Assignment.



(b) Step I.2: Forward Swapping, Min Job-time.



(c) Step I.3: Forward Swapping.



(d) Step II.1: Backward Swapping.

Fig. 4. Heuristics details with graphs.

Fig. 4-b shows the result after Step I.2. C_{max} now is 54 units, which results from job 4 at node 4. i^* is node 4 and $\sigma(i^*) = 4$. k^* is job 4 and $t_{44} = 31$ units. Node 5 succeeds node 4 and it executes job 2. The exchange condition, $t_{44} > min\{t_{4k}, \forall k = \{2\} \quad \sigma(\tau(2)) > \sigma(\tau(4)\}$ and inequalities (19, 20) are satisfied, C_{max} can be further reduced by assigning job 2 to node 4.

Fig. 4-c presents the output after Step I.3. C_{max} now is 53 units, which results from job 2 at node 4. i^* is now node 4 and $\sigma(i^*) = 4$. k^* is job 2 and $t_{42} = 30$ units. Node 3 precedes $i^*(\sigma(3) = \sigma(4) - 1)$. $\Delta_{4,1} = 8$, and inequalities (22–24) are satisfied. By swapping nodes 3 and 4, TS_4 decreases by 1 unit and consequently C_{max} reduces to 52. Both TS_3 and C_3 increase by 11 units. Also, C_5 increases to 52 units. However, $C_3 \leqslant C_4$and $C_5 \leqslant C_4$.

Finally, Fig. 4-d shows the final tour. So, swapping nodes 3 and 4 increases the tour by 8 units, but, C_{max} decreases to 52 units, which results from job 2 being assigned to node 4 or job 4 being assigned to node 5.

References

Adamo, T., Ghiani, G., Guerriero, E., 2020. An enhanced lower bound for the Time-Dependent Travelling Salesman Problem. Comput. Oper. Res. 113, 104795.

Ahonen, H., de Alvarenga, A.G., 2017. Scheduling flexible flow shop with recirculation and machine sequence-dependent processing times: formulation and solution procedures. Int. J. Adv. Manuf. Technol. 89 (1–4), 765–777.

Albiach, J., Sanchis, J.M., Soler, D., 2008. An asymmetric TSP with time windows and with time-dependent travel times and costs: an exact solution through a graph transformation. Eur. J. Oper. Res. 189 (3), 789–802.

Alkaya, A.F., Duman, E., 2013. Application of sequence-dependent traveling salesman problem in printed circuit board assembly. IEEE Trans. Components Packag. Manuf. Technol. 3 (6), 1063–1076.

Arigliano, A., Ghiani, G., Grieco, A., Guerriero, E., Plana, I., 2019. Time-dependent asymmetric traveling salesman problem with time windows: properties and an exact algorithm. Discrete Appl. Math. 261, 28–39.

Averbakh, I., Berman, O., 1996. Routing two-machine flowshop problems on networks with special structure. Transp. Sci. 30 (4), 303–314.

Averbakh, I., Berman, O., Chernykh, I., 2006. The routing open-shop problem on a network: complexity and approximation. Eur. J. Oper. Res. 173 (2), 531–539.

Barbarosoğlu, G., Özdamar, L., Cevik, A., 2002. An interactive approach for hierarchical analysis of helicopter logistics in disaster relief operations. Eur. J. Oper. Res. 140 (1), 118–133.

Basnet, C.B., Foulds, L.R., Wilson, J.M., 2006. Scheduling contractors' farm-to-farm crop harvesting operations. Int. Trans. Oper. Res. 13 (1), 1–15. https://doi.org/10.1111/j.1475-3995.2006.00530.x.

Bays, M.J., Wettergren, T.A., 2015. A solution to the service agent transport problem. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 6443–6450.

Bays, M.J., Wettergren, T.A., 2017. Service agent–transport agent task planning incorporating robust scheduling techniques. Robot. Autonomous Syst. 89, 15–26.

Bektur, G., Saraç, T., 2019. A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. Comput. Oper. Res. 103, 46–63. https://doi.org/10.1016/j.cor.2018.10.010. ISSN 0305-0548.

Bianco, L., Ricciardelli, S., Rinaldi, G., Sassano, A., 1988. Scheduling tasks with sequence-dependent processing times. Naval Res. Logist. 35 (2), 177–184.

Bigras, L.-P., Gamache, M., Savard, G., 2008. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optim. 5 (4), 685–699.

Bretin, A., Desaulniers, G., Rousseau, L.-M., 2020. The traveling salesman problem with time windows in postal services. J. Oper. Res. Soc., 1–15

Cacchiani, V., Contreras-Bolton, C., Toth, P., 2020. Models and algorithms for the traveling salesman problem with time-dependent service times. Eur. J. Oper. Res. 283 (3), 825–843. https://doi.org/10.1016/j.ejor.2019.11.046. ISSN 0377–2217.

Cheng, T.E., Ding, Q., Lin, B.M., 2004. A concise survey of scheduling with time-dependent processing times. Eur. J. Oper. Res. 152 (1), 1–13.

Cheng, B., Lu, H., Xu, X., Shen, W., 2016. Improved local-search-based chaotic discrete particle swarm optimization algorithm for solving traveling salesman problem. J. Comput. Appl. 36, 138–142.

Chernykh, I., Pyatkin, A., 2019. Irreducible bin packing: complexity, solvability and application to the routing open shop. In: International Conference on Learning and Intelligent Optimization. Springer, pp. 106–120.

CPLEX Optimizer, 2019. IBM, (Accessed on 06/06/2019).

Dantzig, G., Fulkerson, R., Johnson, S., 1954. Solution of a large-scale traveling-salesman problem. J. Oper. Res. Soc. Am. 2 (4), 393–410.

Das, S.K., Nagendra, P., 1997. Selection of routes in a flexible manufacturing facility. Int. J. Prod. Econ. 48 (3), 237–247.

Fanjul-Peyro, L., Ruiz, R., Perea, F., 2019. Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. Comput. Oper. Res. 101, 173–182. https://doi.org/10.1016/j.cor.2018.07.007. ISSN 0305-0548.

GAMS, 2019. GAMS – Cutting Edge Modeling, (Accessed on 06/06/2019).

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability, vol. 174, Freeman San Francisco.

Gavish, B., Graves, S.C. The travelling salesman problem and related problems, Working paper OR 0 78–78, MIT Operation Research Center.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R., 1979. Optimization and approximation in deterministic machine scheduling: a survey. Ann. Discrete Math. 5, 287–326.

Gümüş, A.T., Çelik, E. A comprehensive literature review for humanitarian relief logistics in disaster operations management.

Guo, H., Peng, C., 2019. Research on two-stage multi-objective integer programming model of helicopter distribution materials in emergency logistics. In: IOP Conference Series: Materials Science and Engineering, vol. 677, IOP Publishing, 042031.

Hansknecht, C., Joormann, I., Stiller, S. Cuts, primal heuristics, and learning to branch for the time-dependent traveling salesman problem, arXiv preprint arXiv:1805.01415.

Hu, C., Lu, J., Liu, X., Zhang, G., 2018. Robust vehicle routing problem with hard time windows under demand and travel time uncertainty. Comput. Oper. Res. 94, 139–153. https://doi.org/10.1016/j.cor.2018.02.006. ISSN 0305-0548.

Jasim, S.M., Ali, F.H., 2019. Exact and local search methods for solving travelling salesman problem with practical application. Iraqi J. Sci. 60 (5), 1138–1153.

Kim, J.-G., Song, S., Jeong, B., 2019. Minimising total tardiness for the identical parallel machine scheduling problem with splitting jobs and sequence-dependent setup times. Int. J. Prod. Res., 1–16

Lin, Y.-K., Hsieh, F.-Y., 2014. Unrelated parallel machine scheduling with setup times and ready times. Int. J. Prod. Res. 52 (4), 1200–1214.

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

Liu, R., Tao, Y., Xie, X., 2019. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. Comput. Oper. Res. 101. https://doi.org/10.1016/j.cor.2018.08.002. ISSN 0305-0548.

Lo, K.-M., Yi, W.-Y., Wong, P.-K., Leung, K.-S., Leung, Y., Mak, S.-T., 2018. A genetic algorithm with new local operators for multiple traveling salesman problems. Int. J. Comput. Intell. Syst. 11 (1), 692–705.

Logendran, R., McDonell, B., Smucker, B., 2007. Scheduling unrelated parallel machines with sequence-dependent setups. Comput. Oper. Res. 34 (11), 3420–3438.

Mavrovouniotis, M., Müller, F.M., Yang, S., 2016. Ant colony optimization with local search for dynamic traveling salesman problems. IEEE Trans. Cybern. 47 (7), 1743–1756.

Miller, C.E., Tucker, A.W., Zemlin, R.A., 1960. Integer programming formulation of traveling salesman problems. J. ACM 7 (4), 326–329.

Mokotoff, E., 2001. Parallel machine scheduling problems: a survey. Asia-Pacific J. Oper. Res. 18 (2), 193.

Niknamfar, A.H., Niaki, S.T.A., 2016. Soft time-windows for a bi-objective vendor selection problem under a multi-sourcing strategy: binary-continuous differential evolution. Comput. Oper. Res. 76, 43–59. https://doi.org/10.1016/j.cor.2016.06.003. ISSN 0305-0548.

Pfund, M., Fowler, J.W., Gupta, J.N., 2004. A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. J. Chin. Inst. Ind. Eng. 21 (3), 230–241.

Rashidnejad, M., Ebrahimnejad, S., Safari, J., 2018. A bi-objective model of preventive maintenance planning in distributed systems considering vehicle routing problem. Comput. Ind. Eng. 120, 360–381.

Rauchecker, G., Schryen, G., 2019. Using high performance computing for unrelated parallel machine scheduling with sequence-dependent setup times: development and computational evaluation of a parallel branch-and-price algorithm. Comput. Oper. Res. 104, 338–357. https://doi.org/10.1016/j.cor.2018.12.020. ISSN 0305–0548.

Reinelt, G., 1991. TSPLIB – a traveling salesman problem library. ORSA J. Comput. 3 (4), 376–384.

Savelsbergh, 1985. Local search in routing problems with time windows. Ann. Oper. Res. 4 (1), 285–305.

Savelsbergh, An efficient implementation of local search algorithms for constrained routing problems, European Journal of Operational Research 47 (1) (1990) 75–85, ISSN 0377–2217, doi: 10.1016/0377-2217(90)90091-O.

Shi, Y., Boudouh, T., Grunder, O., 2019. A robust optimization for a home health care routing and scheduling problem with consideration of uncertain travel and service times. Transp. Res. Part E 128, 52–95.

Soleimani, H., Ghaderi, H., Tsai, P.-W., Zarbakhshnia, N., Maleki, M., 2020. Scheduling of unrelated parallel machines considering sequence-related setup time, start time-dependent deterioration, position-dependent learning and power consumption minimization. J. Clean. Prod. 249. https://doi.org/10.1016/j.jclepro.2019.119428. 119428, ISSN 0959-6526.

Spieckermann, S., Gutenschwager, K., Voß, S., 2004. A sequential ordering problem in automotive paint shops. Int. J. Prod. Res. 42 (9), 1865–1878.

Tang, H., Miller-Hooks, E., Tomastik, R., 2007. Scheduling technicians for planned maintenance of geographically distributed equipment. Transp. Res. Part E 43 (5), 591–609.

Taş, D., Jabali, O., Woensel, T.V., 2014. A vehicle routing problem with flexible time windows. Comput. Oper. Res. 52, 39–54. https://doi.org/10.1016/j.cor.2014.07.005. ISSN 0305-0548.

Taş, D., Gendreau, M., Jabali, O., Laporte, G., 2016. The traveling salesman problem with time-dependent service times. Eur. J. Oper. Res. 248 (2), 372–383.

UNHCR, 2002. Emergency handbook. http://emergency.unhcr.org/entry/222599/search-and-rescue-response-and-coordination-natural-disasters, (accessed April 15, 2020).

Vallada, E., Ruiz, R., 2011. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. Eur. J. Oper. Res. 211 (3), 612–622. ISSN 0377–2217.https://doi.org/10.1016/j.ejor.2011.01.011.

Van Rossum, G., Drake Jr, F.L., 1995. Python tutorial. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.

Venkatesh, P., Singh, A., Mallipeddi, R., 2019. A multi-start iterated local search algorithm for the maximum scatter traveling salesman problem. In: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE. pp. 1390–1397.

Wang, D., Zhu, J., Wei, X., Cheng, T., Yin, Y., Wang, Y., 2019. Integrated production and multiple trips vehicle routing with time windows and uncertain travel times. Comput. Oper. Res. 103, 1–12.

Wei, X., Qiu, H., Wang, D., Duan, J., Wang, Y., Cheng, T., 2020. An integrated location-routing problem with post-disaster relief distribution. Comput. Ind. Eng. 147, 106632.