Scientia Iranica A (2016) 23(2), 407–419

SCIENTIA IRANICA

Sharif University of Technology
Scientia Iranica
Transactions A: Civil Engineering
www.scientiairanica.com

Parallelization of the branch-and-bound algorithm in transportation discrete network design problem

A. Zarrinmehr*[*] and Y. Shafahi

Department of Civil Engineering, Sharif University of Technology, Tehran, Iran.

Received 1 March 2014; received in revised form 4 June 2015; accepted 25 July 2015

KEYWORDS
Transportation discrete network design;
Parallel computing;
Parallel branch-and-bound algorithm;
Master-slave paradigm.

Abstract. Transportation Discrete Network Design Problem (TDNDP) aims at choosing a subset of proposed projects to minimize the users' total travel time with respect to a budget constraint. Because TDNDP is a hard combinatorial problem, recent research has widely addressed heuristic approaches and ignored the exact solution. This paper explores how application of parallel computation can affect the performance of an exact algorithm in TDNDP. First, we show that the Branch-and-Bound (B&B) algorithm, proposed by LeBlanc, is well adapted to a parallel design with synchronized Master-Slave (MS) paradigm. Then, we develop a parallel B&B algorithm and implement it with two search strategies of Depth-First-Search (DFS) and Best-First-Search (BFS). Detailed results over up to 16 processing cores are reported and discussed in an illustrative example of the Chicago Sketch network. The results suggest an almost linear speedup for both strategies, which slightly drops as more processing cores are added. When using 16 processing cores, the speedup values of 11.80 and 12.20 are achieved for DFS and BFS strategies, respectively. Furthermore, the BFS strategy reveals a very fast parallel performance by finding the optimal solution via the minimum computational effort.

1. Introduction

Transportation Discrete Network Design Problem (TDNDP) emerges in transportation planning as an urban infrastructural decision-making problem. It targets the construction of new projects, i.e. roads, in an urban roads network so as to minimize the total travel time of network users, while holding the budget constraint. Considering a binary decision variable for each project (whether the project is decided to be constructed or not), TDNDP will be a combinatorial problem with a binary search space, which exponentially enlarges as the number of projects increases. This is why the main body of related research has neglected the exact solutions of the problem and addressed the problem through heuristic approaches.

The common idea in all heuristic approaches is to achieve a rather high-quality feasible solution in a reasonable amount of time [1]. The main drawback in the application of heuristic algorithms is that they do not provide insight into the quality of the achieved solution. This is because the exact solution to the problem, as a benchmark, is not available in large instances. Achieving the exact solution, however, is a computing intensive problem, which may require more than 1-year CPU-time, even in moderate TDNDPs [1]. As a result, it is important to explore how the application of parallel computation can contribute to speed up extraction of an exact solution.

Using parallel computing, this article reinvestigates the exact solution of TDNDP as a well-known transportation problem. To do so, the conventional Branch-and-Bound (B&B) algorithm, introduced in

*. *Corresponding author. Tel.: +98 21 82883386; Fax: +98 21 82884914*
E-mail addresses: amirali.zarrinmehr@modares.ac.ir (A. Zarrinmehr); shafahi@sharif.edu (Y. Shafahi)

the early study of LeBlanc [2], will be considered with two standard search strategies of Depth-First-Search (DFS) and Best-Fist-Search (BFS). The algorithm is parallelized using a Master-Salve (MS) paradigm. Application of the algorithm to the Chicago Sketch transportation network with 12 proposed projects supports that adding more processing cores (up to 16 cores in this study) promises a notable reduction in computation time of the results.

The rest of this paper is organized as follows. Section 2 overviews the background research. Section 3 formally describes TDNDP in a bi-level mathematical programming formulation. Some prerequisite concepts about B&B algorithms and their parallelization will be introduced in Section 4. The application of a synchronized MS parallel design to the B&B algorithm in TDNDP is discussed in Section 5, where the parallel algorithm is presented and some implementation notes are added. The case study is introduced in Section 6 and computation results are reported and further discussed. The paper is concluded and remarks for future work are added in Sections 7 and 8.

2. Related research

As a combinatorial problem, TDNDP falls in the category of NP-hard complexity class of network design problems [3]. Combinatorial explosion of such a problem leads to an intractable running-time, which is the main barrier for the application of exact algorithms. As a result, exact solution methods in TDNDP date back only to a few early studies which were not further developed due to their computationally demanding nature [4].

In NP-hard problems like TDNDP, there are various approaches, which can assist in finding a solution in a reasonable running-time. One approach is the application of heuristics, which is, to trade off the quality of the found solution against the corresponding running-time. In TDNDP, this has been the dominant approach for which intense literature is available [1,5]. However, application of parallel computation is another approach which harnesses multiple computation resources to alleviate the computational burden of the problem [6]. A short overview of both approaches is given here.

There have been a variety of approaches, developed in TDNDP literature, to heuristically trade off the precision of the solution against the computation time [1]. Some early studies manipulated the problem formulation and obtained approximated solutions for the problem [4], while others introduced decomposition techniques to overcome the problem size [7,8]. Meta-heuristics, as novel heuristics, have also been extensively used and investigated in recent literature. Various meta-heuristic algorithms have been developed and compared in TDNDP in categories of genetic, ant colony optimization, simulated annealing, and particle swarm optimization. The interested reader may find more on this topic in a recent review paper [4] and the references therein.

Heuristics and meta-heuristics are promising approaches to evade the exponential complexity of TDNDPs. Although these approaches can reach an appropriate solution in many instances, they do not provide insight into the goodness of their solution. As a result, exact solution methods always remain worth not only to find the best solution of the problem, but also even to test and validate heuristic approaches.

In recent decades, the advent of parallel computing facilities encouraged the exact solution of many NP-hard problems. Promising results have been reported in the literature for many of such problems [9]. Parallel algorithms, especially parallel Branch-and-Bound (B&B) algorithms, as a result, were organized in early 1980's. These algorithms were further developed during the last two decades and in various combinatorial problems [6]. The results were impressive in finding the exact solution to some problems which had been considered unsolvable for decades. Asymmetric traveling salesman problems with thousands of cities [10], as well as instances of the quadratic assignment problem with hundreds of variables [11], are among the problems for which the exact solutions were found by devising parallel branch-and-bound algorithms. Interested reader can find more information in [6,9].

In transportation planning discipline, in spite of the demanding computational nature of many problems, parallel computing has not received much attention so far. Parallelization of the transportation network design problem, in either public transit or urban transportation network design problem, has been the topic of limited research. In case of public transit network design, no work, to the best of our knowledge, has been dedicated to addressing a parallel exact solution. The existing contributions are limited to the application of parallel meta-heuristics to the problem. Agrawal and Mathew [12] suggested a parallel genetic algorithm for the problem of transit routes network design, while minimizing the total system costs. They suggested an inherent parallelization scheme for the genetic algorithm, in which the fitness function was evaluated, concurrently, for the solutions. Performance measures were reported for two parallel models of global message passing interface and global parallel virtual machine. It was observed that the global parallel virtual machine model would outperform the other one. Yan et al. [13] proposed a model based on parallelization of the ant colony algorithm for maximizing the number of direct travelers per unit length in a bus network design problem. Parallel performance measures, however, have not been reported in their study.

In a recent paper, Cooper et al. [14] implemented a parallel multi-objective model based on the genetic algorithm for minimizing both users' average travel time and the operator's cost. They suggested application of an islands approach with a random migration policy to improve the performance of their parallel algorithm. In their approach, the population of the genetic algorithm is split into islands (sub-populations), among which randomly selected solutions are exchanged. For a transit network with 127 bus stops, when using 24, 48, and 96 processing cores, they report speedup values of 17.8, 30.1, and 37.0, respectively.

Parallel solutions for urban transportation network design problem have also been the topic of research in Zarrinmehr [15] and Zarrinmehr and Shafahi [16]. An exact B&B algorithm was undertaken in both of these studies. Zarrinmehr [15] reported the parallel performance of a synchronized MS parallel B&B algorithm in a case study of the Chicago Sketch transportation network. While addressing users' route choice behavior through *system-optimal flows*, he reported that by using 16 processing cores, the parallel B&B algorithms can achieve speedup values of 10.85 and 9.86 in cases of BFS and DFS search strategies, respectively. Zarrinmehr and Shafahi [16] investigated how the performance of a parallel B&B algorithm with DFS strategy in TDNDP could be accelerated. They suggested assigning greedy solutions to the idle processors at the start of the B&B algorithm. However, authors reported the performance of their algorithm in terms of the total number of parallel iterations rather than the real running-times. Their results suggest a potential super-linear speedup in four illustrative examples of the Sioux-Falls transportation network.

The real performances of parallel algorithms for rather large transportation networks in TDNDP, to our knowledge, have not been addressed so far in the literature. This paper explores how parallelization can contribute to speed up the solution of a TDNDP. Figure 1 shows how the research roadmap in this study is narrowed down to find an exact solution for this problem. As shown in this figure, the paper will address an exact solution for the problem. To achieve that, the paper targets a parallel B&B algorithm based on the algorithm proposed by LeBlanc [2]. Further, the corresponding paradigm in this paper will be a synchronous MS parallelization, in which one processor will hold the main information, generate new tasks, and distribute them across working processors at predefined intervals.

3. Formal description of TDNDP

TDNDP is often known as a problem with a bi-level programming formulation. At the upper level, it seeks an optimum subset of a set of proposed projects (i.e., roads) in order to minimize the users' total travel time in the network, while at the lower level, it solves a Traffic Assignment Problem (TAP) [1]. To formally describe the problem, we use the following notations:

V	The set of network vertices;
A	Set of network links;
$N(V,A)$	Current network, which is assumed to be connected;
A_y	Set of proposed projects (links) to be added to the current network;
P	Set of Origin-Destinations (ODs), $P \subseteq A \times A$;
d_{rs}	The travel demand from r to s, $(r,s) \in P$;
y_a	Binary decision variable corresponding to project a, $a \in A_y$, taking values 1 or 0, indicating whether to construct a project or not;
y	Binary vector of decision variables;
K_{rs}	The non-empty set of different paths form r to s, $(r,s) \in P$;
$A(y)$	Set of network links after decision y has been made, $A(y) = A \cup \{a \in A_y : y_a = 1\}$;
$K_{rs}(y)$	The non-empty set of different paths form r to s after decision y has been made, $(r,s) \in P$;
f_k	Amount of flow in path k from r to s, $k \in K_{rs}(y)$;
δ_{ak}	Binary variable taking values 1 or 0 indicating whether link "a" belongs to path k or not, $a \in A(y)$, $k \in K_{rs}(y)$;

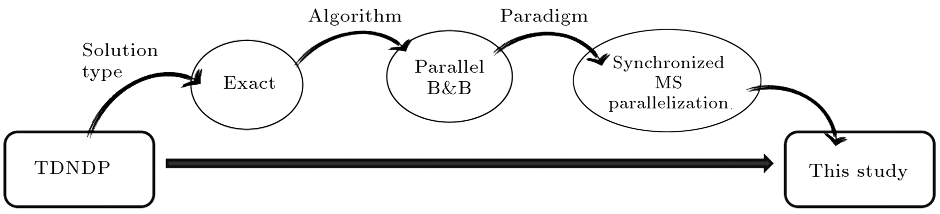


Figure 1. The scope of this study.

x_a	Amount of flow in link "a", $a \in A(y)$;
$x(y)$	Vector of traffic flow on network links after decision y has been made;
$t_a(x_a)$	Volume-delay function of link a, $a \in A(y)$, assumed to be convex and differentiable for $x_a \geq 0$;
c_a	The cost related to construction of project a, $a \in A_y$;
B	Available budget for the construction of given projects.

The Upper Level Problem (ULP) of TDNDP, now, can be written as follows:

$$\text{Min}_y T(y) = \sum_{a \in A(y)} x_a t_a(x_a), \tag{1}$$

$$\text{s.t.}: \quad \sum_{a \in A_y} c_a y_a \leq B, \tag{2}$$

$$y_a = 0 \text{ or } 1, \quad \forall a \in A_y, \tag{3}$$

$$x(y): \text{ user equilibrium flow corresponding to decision vector } y. \tag{4}$$

The User Equilibrium (UE) in the ULP is a situation in which for any OD, all paths have an equal travel time which is not greater than those of unused paths [17]. This situation is resulted by solving a TAP, which can be formulated in the Lower-Level Problem (LLP):

$$[\text{LLP}(y)] \qquad \text{Min} \sum_{a \in A(y)} \int_{u=0}^{x_a} t_a(u)du, \tag{5}$$

$$\text{s.t.}: \quad \sum_{k \in K_{rs}(y)} f_k = d_{rs}, \quad \forall (r,s) \in P, \tag{6}$$

$$f_k \geq 0, \quad \forall k \in K_{rs}(y), \quad \forall (r,s) \in P, \tag{7}$$

$$x_a = \sum_{(r,s) \in P} \sum_{k \in K_{rs}(y)} \delta_{ak} f_k, \quad \forall a \in A(y). \tag{8}$$

Henceforth, the above formulation for TAP will be referred to as a UE-type TAP. In transportation literature, TAP commonly stands for a UE-type problem. However, while addressing the B&B algorithm of LeBlanc [2], we will also need another kind of TAP that is formulated to find System Optimal (SO) flows, namely a SO-type TAP. The SO-type TAP differs from the UE-type one only in its objective function. It can be written as follows [17]:

$$\text{Min}_y T(y) = \sum_{a \in A(y)} x_a t_a(x_a),$$

$$\text{s.t.}: \quad (6), (7), (8). \tag{9}$$

4. Parallel B&B algorithms in combinatorial problems

Parallelization is a way to speed up B&B algorithms in order to find the exact solution of combinatorial problems. Although parallelization cannot cope with the combinatorial nature of NP-hard problems, the experience of the past two decades suggests that it can help solving moderate and rather large problems that previously seemed to be unsolvable [6,9,18]. This section first briefly describes the two main search strategies of a B&B algorithm (i.e., BFS and DFS) and, then, introduces some prerequisite concepts about parallel B&B algorithms.

4.1. B&B search strategies

There are two main strategies to traverse a B&B tree, often known as BFS and DFS. These strategies arise from the selection rule, with which one node is selected and removed out of active nodes. If the selection priority is given to a node with the least lower bound value, the strategy of BFS emerges. Also, if the deepest node in the B&B tree is given the priority, the corresponding strategy would be called DFS [18]. To investigate the performance of different strategies, researchers categorized nodes of the B&B tree (partial solutions) into three distinct sets [6,9,18]. Let f^* be the optimal objective function of a combinatorial problem; then, categories are as follow:

- Critical nodes: Those nodes with lower bound values smaller than f^*;
- Undecidable nodes: Those nodes with lower bound values equal to f^*;
- Eliminable nodes: Those nodes with lower bound values greater than f^*.

The minimal subset of the B&B tree that must be traversed in order to find the optimal solution and prove its optimality, called the minimal tree [6,18]. The minimal tree includes all critical nodes and maybe some undecidable nodes. There is no guarantee that any B&B algorithm restricts its search to the minimal tree, because the traversed tree is created dynamically in the course of the algorithm and it has an irregular structure which is not known at the outset [6].

The BFS strategy focuses the search on critical and undecidable nodes and commonly reveals a robust performance. The main drawback of the BFS strategy is its required memory which may grow exponentially in the worst case. On the other hand, the DFS strategy may evaluate lots of eliminable nodes and become more time-consuming, but it requires much less memory and can be effective in dealing with large-size problems. Finally, using a proper strategy always remains dependent to the problem at hand and available resources [6,18].

4.2. Parallelization of B&B algorithms

There are various approaches to search a B&B tree in parallel. In a node-based approach, the related computation at each node of the B&B tree (e.g., lower bound calculation) is parallelized over processors. A tree-based approach, on the other hand, is one of the most famous approaches in which processors work on different parts of the tree, simultaneously. Tree-based approaches may have interesting effects on the behavior of the algorithm and, therefore, they have been widely studied by researchers [18]. Through all various tree-based approaches, this paper focuses on a special topic, namely MS paradigm; but, beforehand, some quantitative measures in parallel computing should be introduced.

Assume that $T(1)$ and $T(p)$ are running-times of a program when it is run on 1 and p processors, respectively. Then, $S(p)$ as the speedup of parallelization with p processors is defined as follows [19]:

$$S(p) = \frac{T(1)}{T(p)}. \tag{10}$$

Speedup is an indicator of how many times the program would run faster due to parallelization. According to Amdahl's law in parallel computing, it can be written that [19]:

$$1 < S(p) < p. \tag{11}$$

A linear speedup is a speedup that almost equals the number of processors. Achieving a linear speedup shows an effective parallelization.

In parallel B&B algorithms, Amdahl's law, however, is not always held. This behavior is often known as an "anomaly" [20]. A super-linear speedup is a kind of anomaly in which the speedup exceeds the number of processors. Super-linear speedup arises due to parallelization in B&B algorithms when an early access of processors to a good solution brings about deletion of further active nodes and reduction of the search in the B&B tree [21].

Studying speedup bounds is not trivial in parallel B&B algorithms due to complexities and non-deterministic behaviors. Nevertheless, researchers have studied, theoretically, a few parallel paradigms. A synchronized MS paradigm is one of these paradigms, which can be briefly described as follows [18]: A main processor, namely the master, holds the important information, e.g. the B&B tree, the incumbent, etc. The master distributes active nodes as jobs among other processors, namely the slaves (which is a MS paradigm). The slaves receive the nodes, evaluate related bounds, and return the results to the master at predefined intervals (which is a synchronized paradigm). The master then updates its information and generates new jobs if needed.

The theoretical study of synchronized MS paradigm was accompanied with 2 assumptions for analysis simplification:

- The master-slave data exchange is done in a zero time;
- All slaves evaluate received nodes in a uniform amount of time (known as grain-size uniformity).

With the above assumptions, if $I(1)$ and $I(p)$ stand for the number of B&B iterations with 1 and p processors, respectively, then the real-case speedup, $S(p)$, can be estimated by the theoretical speedup, $S^t(p)$, as follows [20,22,23]:

$$S(p) \simeq S^t(p) = \frac{I(1)}{I(p)}. \tag{12}$$

Theoretical studies of anomalous behaviors in parallel B&B algorithms with MS paradigm were organized in the middle of 80s. These studies proved that anomalous behaviors stem from the ambiguity in the selection of undecidable nodes when they emerge as selection ties. As a result, one can say that Amdahl's law (previously presented in Inequality (11)) is theoretically held in B&B trees with empty set of undecidable nodes.

5. Methodology

This section discusses how a synchronized MS parallelization paradigm is adapted for parallelization of the B&B algorithm in TDNDP and presents a flowchart and corresponding implementation notes for the parallel algorithm. But we require, beforehand, a quick overview of LeBlanc's sequential B&B algorithm, to which will be referred in next subsections.

5.1. The sequential algorithm

As previously mentioned, this paper addresses LeBlanc's B&B algorithm as the basic sequential algorithm to be parallelized. To exactly solve TDNDP, LeBlanc organized a B&B algorithm for TDNDP, in which the given projects were decided whether to be constructed or not at subsequent levels of a binary tree. Assuming n to be the number of given projects, the B&B tree, introduced by LeBlanc, is a rooted binary tree with n levels in which, at the root, all projects are initially undecided. The decision whether to construct project i or not $(i = 1, 2, ..., n)$ is made at level i of the tree. Figure 2 shows the structure of the tree, in which each node is coded by a string of n binary variables. Symbols 1, 0, or - at the ith place respectively indicate that the project i is decided whether to be constructed, not to be constructed, or yet undecided (as what is the case for partial solutions at levels 1 to $n - 1$).

LeBlanc's B&B algorithm iteratively decomposes partial solutions and when a budget-wise feasible and

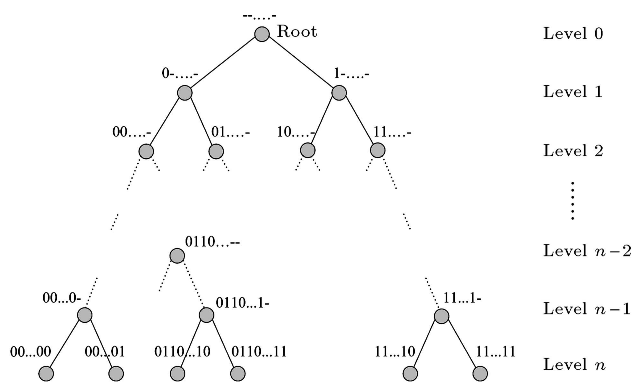


Figure 2. The structure of the B&B tree in TDNDP in LeBlanc's algorithm.

complete solution emerges, solves a UE-type TAP to evaluate it [2]. To achieve a lower bound on partial solutions, LeBlanc also relaxed Constraints (2) and (4), added Constraints (6), (7), and (8) to the ULP; and proved that the objective function of the emerging problem would work as a lower bound. This lower bound can be calculated by solving a SO-type TAP in which all undecided projects are assumed to be constructed [2]. Figure 3 provides a flowchart for LeBlanc's B&B algorithm. In this figure, the term node stands for a B&B tree node and the terms right and left children are defined in the same way as that shown in Figure 2.

5.2. Application of a synchronized MS parallelization paradigm on the problem

A synchronized MS paradigm of parallelization well adapts LeBlanc's B&B algorithm in TDNDP. Figure 4 shows this paradigm in a diagram. According to the diagram, the master processor holds the main information of the B&B algorithm. Each parallel iteration starts with generating new nodes to be evaluated. The slave processors that have already maintained the network information receive the newly generated nodes in the form of binary string codes. Each slave processor decodes the received binary string, performs a traffic assignment on the corresponding network, and sends its evaluation back to the master. The parallel iteration finishes as the master receives the evaluation data and updates the incumbent and active nodes.

Choosing an appropriate parallelization paradigm for a problem is highly affected by both communication and idling overheads in the parallel program [19]. Two main reasons, in this paper, support the application of a synchronized MS paradigm:

- Communication overhead due to a centralized paradigm is negligible;

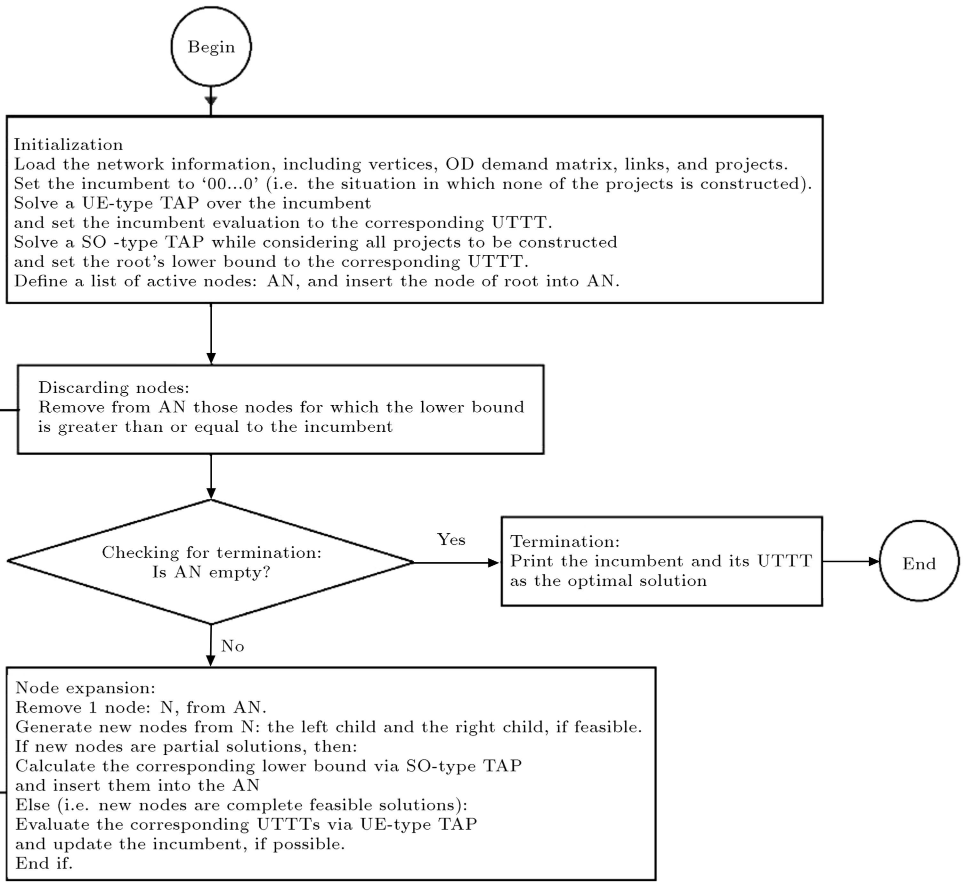

Begin
Initialization
Load the network information, including vertices, OD demand matrix, links, and projects.
Set the incumbent to '00...0' (i.e. the situation in which none of the projects is constructed).
Solve a UE-type TAP over the incumbent
and set the incumbent evaluation to the corresponding UTTT.
Solve a SO -type TAP while considering all projects to be constructed
and set the root's lower bound to the corresponding UTTT.
Define a list of active nodes: AN, and insert the node of root into AN.
Discarding nodes:
Remove from AN those nodes for which the lower bound
is greater than or equal to the incumbent
Checking for termination:
Is AN empty?
Yes
Termination:
Print the incumbent and its UTTT
as the optimal solution
End
No
Node expansion:
Remove 1 node: N, from AN.
Generate new nodes from N: the left child and the right child, if feasible.
If new nodes are partial solutions, then:
Calculate the corresponding lower bound via SO-type TAP
and insert them into the AN
Else (i.e. new nodes are complete feasible solutions):
Evaluate the corresponding UTTTs via UE-type TAP
and update the incumbent, if possible.
End if.

Figure 3. Flowchart for LeBlanc's B&B algorithm.

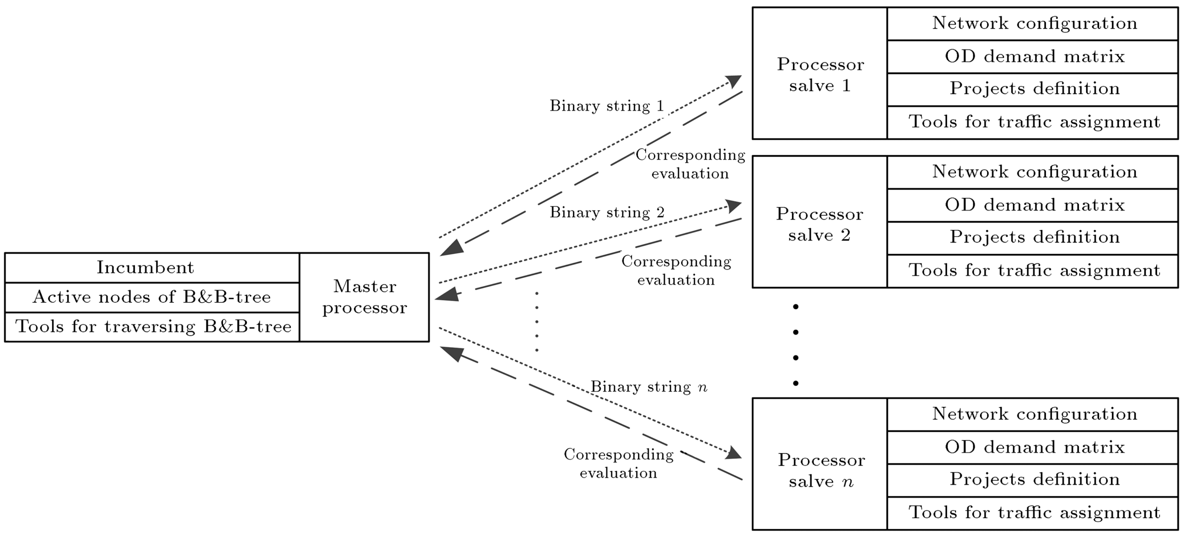


Figure 4. Diagram of parallelization.

- Idling overhead due to a synchronized paradigm is not much.

5.2.1. Communication overhead

In parallel computing, the communication between processors usually results in communication overhead. This is the case particularly for MS paradigms when the access to the master may become a bottleneck [18]. Communication overhead is totally dependent on the task grain size of processors and the amount of data exchanged between them [19]. The parallel B&B algorithm in this paper has a coarse-grain structure, because processors will solve a TAP at each parallel iteration, which is a rather time-consuming procedure.

Furthermore, data exchange between processors will be restricted to only sending and receiving the binary strings and related evaluations at each iteration. To achieve this, all the required information (network, demand matrix, and projects) will initially be loaded over slave processors. Afterwards, each slave processor receives a binary string as a solution to be evaluated, adjusts the underlying network with the received information, solves a TAP over that, and sends its evaluated value back to the master.

5.2.2. Idling overhead

Idling overhead often takes place in synchronized parallelization when processors must communicate at predefined intervals and idling time may be incurred to some processors due to synchronization [18]. Uniformity of the grain-size, therefore, plays an important role in alleviating idling overhead. In the parallel B&B algorithm used in this paper, we assign each processor to solve one TAP (i.e., evaluate one node) at each iteration. The computational complexity of TAPs mainly depends on the network size and configuration, which remains almost unchanged in the real networks by

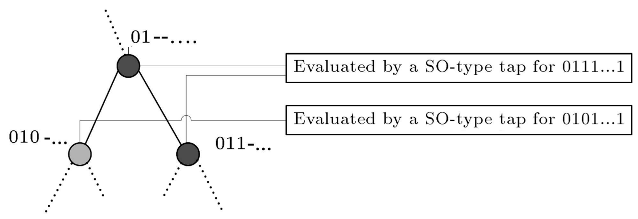

01--....
Evaluated by a SO-type tap for 0111...1
Evaluated by a SO-type tap for 0101...1
010-...
011-...

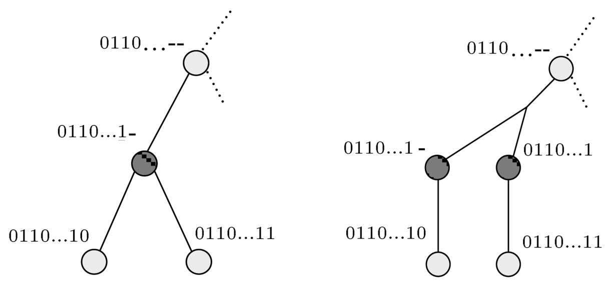
Figure 5. Lower bound evaluation in the B&B tree.

adding or deleting a couple of projects. However, there still remains another problem to parallelize LeBlanc's algorithm with a uniform grain-size.

Let us define a node expansion in LeBlanc's B&B tree as creation and evaluation of feasible children of one node. Then, the potential problem occurs when there is a difference between expanding nodes of levels $1, 2, ..., n-2$ and those of level $n-1$. For nodes of levels $1, 2, ..., n-2$, the expansion can be done by exactly solving one TAP as shown in Figure 5. This is because after assuming undecided projects to be constructed (required for lower bound calculation), the right child, which takes the new decision of construction, will have the same lower bound as that of its father and needs no further bound calculation. Therefore, expansion of such nodes is computationally equivalent to solving one TAP only for the left child.

The expansion of nodes of level $n-1$, however, may require solving two TAPs rather than one. Both children of those nodes, if feasible, are complete solutions for which a UE-type TAP must be solved for evaluation. This problem opposes the uniformity of the grain-size in our parallel algorithm, but can be tackled by a simple modification in the B&B tree, shown in Figure 6.

According to the modification shown in Figure 6, for any node of level $n-1$, if the right child thereof is also feasible (Figure 6(a)), we duplicate the nodes each



(a) Before modification (b) After modification

Figure 6. The B&B tree at level $n-1$.

for one child (Figure 6(b)). The priority is then greedily given to the right child with one more constructed project. Therefore, in the modified tree, expansion of the nodes of level $n-1$, as well as the nodes of levels $1, 2, ..., n-2$, will computationally equal to one TAP solution and at the same time, the grain-size uniformity of parallel algorithm is also preserved.

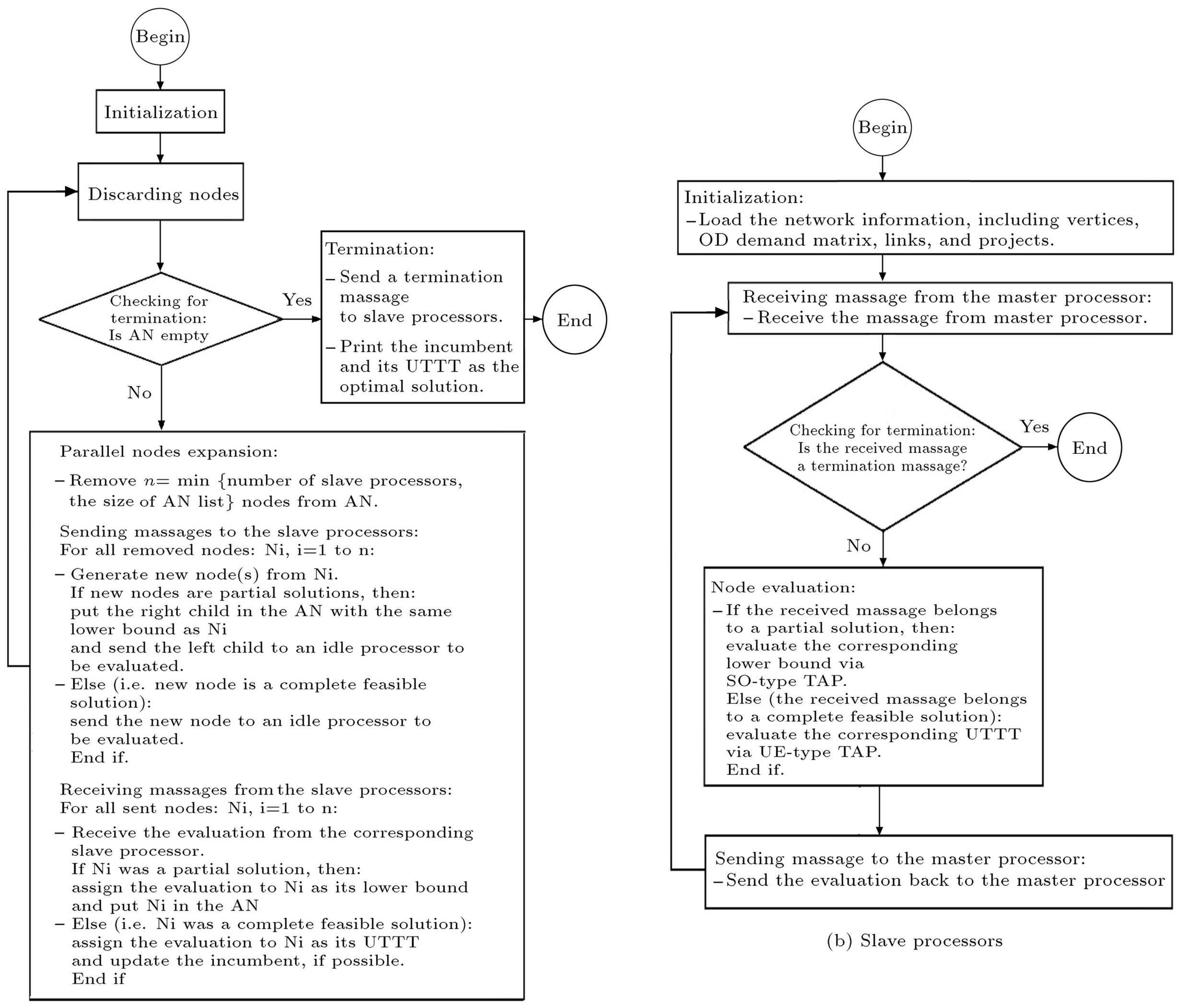
5.3. Parallelization of the sequential algorithm

The focus of this paper in parallelization of LeBlanc's B&B algorithm is on the stage of node expansion, namely parallel nodes expansion. The parallel B&B algorithm can be viewed from two points: the master processor and the slaves. Figure 7 represents flowcharts of the parallel algorithm for both master and slaves. Most steps in this figure are similar to those in Figure 3. The main differences are at the stages of parallel nodes expansion, slaves receiving massages and working on them, and the termination part where a parallel feature is applied to the algorithm.

5.4. Implementation notes

Here, we add some implementation notes about the algorithm:

- The parallel algorithm was implemented in Java programming language and the library MPJ was exploited for parallelization;



Termination:
- Send a termination massage to slave processors.
- Print the incumbent and its UTTT as the optimal solution.

Parallel nodes expansion:
- Remove n= min {number of slave processors, the size of AN list} nodes from AN.

Sending massages to the slave processors:
For all removed nodes: Ni, i=1 to n:
- Generate new node(s) from Ni.
  If new nodes are partial solutions, then:
  put the right child in the AN with the same lower bound as Ni
  and send the left child to an idle processor to be evaluated.
- Else (i.e. new node is a complete feasible solution):
  send the new node to an idle processor to be evaluated.
  End if.

Receiving massages from the slave processors:
For all sent nodes: Ni, i=1 to n:
- Receive the evaluation from the corresponding slave processor.
  If Ni was a partial solution, then:
  assign the evaluation to Ni as its lower bound and put Ni in the AN
- Else (i.e. Ni was a complete feasible solution):
  assign the evaluation to Ni as its UTTT
  and update the incumbent, if possible.
  End if

Initialization:
- Load the network information, including vertices, OD demand matrix, links, and projects.

Receiving massage from the master processor:
- Receive the massage from master processor.

Checking for termination: Is the received massage a termination massage?

Node evaluation:
- If the received massage belongs to a partial solution, then:
  evaluate the corresponding lower bound via SO-type TAP.
  Else (the received massage belongs to a complete feasible solution):
  evaluate the corresponding UTTT via UE-type TAP.
  End if.

Sending massage to the master processor:
- Send the evaluation back to the master processor

(a) Master processor (b) Slave processors

Figure 7. Flowcharts for the parallel B&B algorithm.

Table 1. Definition of projects in the Chicago Sketch network.

Project number	From vertex	To vertex	$fftt_a$ (minutes)	c_a (vehicles/hour)	b_a (units of budget)
1	29	28	3.52	12000	16
2	3	5	3.73	12000	16
3	397	403	4.65	12000	17
4	605	399	5.37	12000	18
5	362	360	7.20	12000	20
6	407	681	7.55	12000	22
7	550	564	7.73	12000	22
8	580	568	10.10	12000	22
9	766	779	10.61	12000	23
10	42	50	14.20	12000	26
11	250	217	17.64	12000	29
12	527	529	18.03	12000	33

- TAPs were solved using the convex combination algorithm [17];
- The B&B algorithm was implemented with two strategies of DFS and BFS. This was done in the master processor by giving the corresponding selection priorities when a node was selected and removed from active nodes;
- Since the TAP brought about a coarse-grain parallelization, especially for real-life transportation networks, the master would experience a notable idling overhead. Therefore, the master was also designated to solve a TAP in conjunction with slaves;
- The program was finally run on two clusters, each with 2 CPUs of type Intel Xeon E5504 2.00 GHz. With 4 processing cores at each CPU, the total number of 16 processing cores were available to parallelize the program.

6. Computational performance

6.1. Case study definition

The case study of this paper, over which parallel performance and results of the algorithm are reported, is the Chicago Sketch network. The Chicago Sketch network is an aggregated, but yet a rather large, representation of the Chicago region network [24]. It contains 933 vertices, 2950 links, and 93513 non-zero trip demand values in the OD matrix. The data for demand in the Chicago Sketch network provides very low levels of congestion, which may not be naturally subjected to a TDNDP. The overall demand matrix, as a result, is initially doubled (suggested in Bar-Gera [24]) to make the underlying network more congested and realistic for design applications.

In order to introduce a set of projects to select from, 12 artificial projects were proposed following the pattern of network links. The volume-delay functions for projects, as well as network links, follow the traditional BPR format:

$$t_a(x_a) = fftt_a\left(1 + 0.15\left(\frac{x_a}{c_a}\right)^4\right), \qquad (13)$$

in which:

$fftt_a$	is the free-flow travel time on project a (minutes),
c_a	is the practical capacity of project a (vehicles/hour),
$t_a(x_a)$	is the travel time on project a (minutes).

Table 1 presents detailed information of the projects based on the BPR function. In this table, b_a also stands for the budget (units of budget) required to construct project a. The total available budget is assumed to be 100 (units of budget), which is a mid-size budget level. Although the capacity of 12000 (vehicles/hour) sounds rather large for the projects, it is not yet beyond the capacity range of the existing links of the Chicago Sketch network. In this network, there are more than 30 non-connector links, for which the practical capacity is 12000 or even more vehicles/hour [24]. Addressing projects with low-capacities, however, may require application of more accurate TAP solution algorithms compared to the classic convex-combination algorithm (Frank-Wolfe algorithm) used in this paper [25].

6.2. Numerical results

Based on the parallel synchronized MS algorithm previously introduced, the program was run with two strategies of DFS and BFS. Related results for 1 to 16

Table 2. Parallel computation results.

(a) DFS strategy

Number of processing cores[1], p	Total number of evaluated nodes	Number of evaluated nodes of type			Total number of parallel iterations, $I(p)$	Running-time, $T(p)$ (minutes)
		Critical	Undecidable	Eliminable		
1	1013	767	0	246	1013	271.71
2	1013	767	0	246	508	144.83
4	1019	767	0	252	258	76.49
8	1029	767	0	262	132	42.57
16	1062	767	0	295	69	23.02

(b) BFS strategy

Number of processing cores[1], p	Total number of evaluated nodes	Number of evaluated nodes of type			Total number of parallel iterations, $I(p)$	Running-time, $T(p)$ (minutes)
		Critical	Undecidable	Eliminable		
1	767	767	0	0	767	208.09
2	767	767	0	0	384	110.08
4	767	767	0	0	193	56.71
8	767	767	0	0	99	31.39
16	767	767	0	0	53	17.06

[1]This number includes the master processor as well, as previously mentioned in Subsection 5.4.

Table 3. Maximum number of active nodes for strategies of DFS and BFS.

Number of processing cores, p	The maximum number of active nodes	
	DFS	BFS
1	6	232
2	12	230
4	22	229
8	40	229
16	76	230

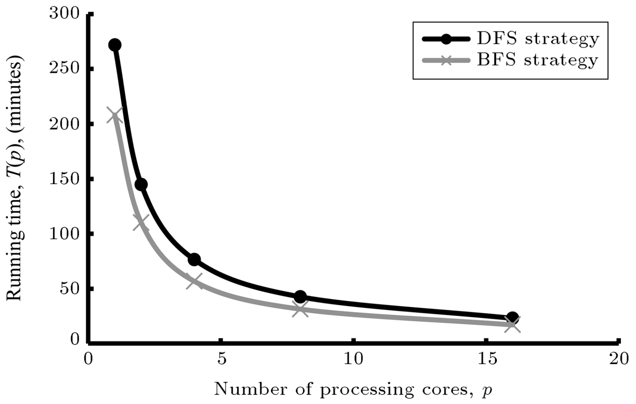


Figure 8. The running-time of parallel programs.

processing cores are shown in Table 2. It must be noted that the data is not known in advance and it will be extracted after running the programs.

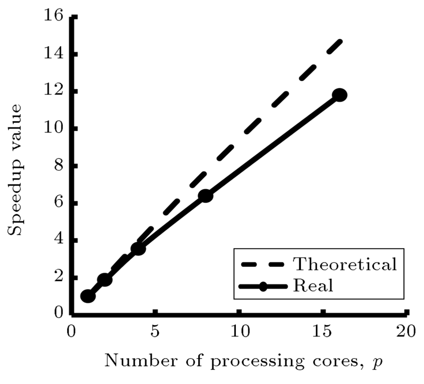
The required memory for both strategies of DFS and BFS can also be estimated by the maximum number of active nodes during the program execution, as represented in Table 3.

After running the program, the optimal solution, as a binary string of length 12, was found [000100111000] with the objective function of 65045456 vehicle-minutes.

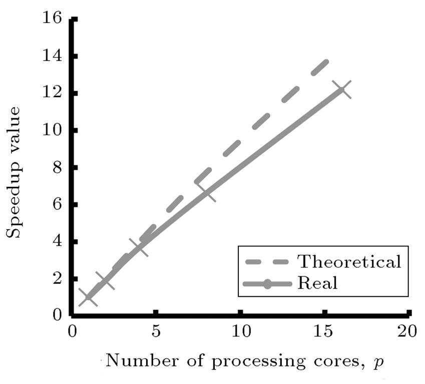
6.3. Speedup, anomalous behavior, and memory analysis

Based on the results of Table 2, the program with the BFS strategy reveals a notably faster performance than that of the strategy DFS. As Table 2(a) suggests, there are many eliminable nodes in the traversal of DFS strategy. As a result, much more parallel iterations in the DFS strategy are required to evaluate these extra nodes. According to Table 2(b), the number of eliminable nodes for the BFS strategy remains zero even when the number of processors increases to 16. This is why the strategy BFS reveals a notably faster performance than the strategy DFS, as shown in Figure 8, in which running-time curves are plotted for both strategies.

As previously discussed in theoretical studies about parallel B&B algorithms, anomalous behaviors (that contradict with Amdahl's law) may occur when there are selection ties over undecidable nodes in the B&B tree [20]. In our TDNDP instance, however, Table 2 suggests that there is no undecidable node for both parallel strategies of BFS and DFS. As a result, Amdahl's law theoretically holds for the synchronized



(a) Parallel DFS program



(b) Parallel BFS program

Figure 9. Speedup curves in the Chicago Sketch case study.

MS parallelization of the B&B algorithm in TDNDP instance of this paper.

Confining the search to the minimal B&B tree is other interesting information that can be extracted from Table 2(b) for the BFS strategy. As shown in this table, the BFS strategy traverses neither of the eliminable nodes nor the undecidable ones. This clearly adapts to what has been previously introduced as the minimal tree in B&B algorithms. It is noteworthy that even for 16 parallel processing cores, this behavior in achieving the solution through the minimum computational effort is consistent.

The required memory for strategies DFS and BFS can be compared using Table 3. The required memory for the DFS strategy is much lower than that for the BFS. It increases linearly at first, but slightly drops for higher values of p. For the BFS strategy, this behavior seems quite different. The required memory for this strategy is rather high, but it remains almost unchanged by addition of more processing cores.

Speedup curves are the other information. The real speedup, $S(p)$, and theoretical speedup, $S^t(p)$, (introduced, respectively, in Eqs. (10) and (12)) are calculated in Table 4. Corresponding curves for strategies DFS and BFS are also shown in Figure 9. This figure indicates that real speedup values are lower than the theoretical ones. This gap is mainly due to a synchronized design, in which a short lag in one processing core is incurred to all of them as an idling overhead. Although the gap between real and theoretical speedups, shown in Figure 9, slightly increases for higher values of p, both strategies, DFS and BFS, reveal a notable increment in their real speedups by adding more processors up to 16 processing cores in this study. Both strategies achieve a real speedup of almost 12 (12.20 and 11.80 for BFS and DFS, respectively) when 16 processing cores are used for parallelization. These are slightly more than, but still similar to, the speedup values reported by Zarrinmehr [15] (10.85 for BFS and 9.86 for DFS strategies) who solved the problem with a normal demand (not doubled as in this study) and a *system-optimal flow* pattern.

Table 4. Speedup results for the strategies of DFS and BFS in the Chicago Sketch case study.

Number of processing cores, p	**DFS speedup values**		**BFS speedup values**	
	$S^t(p)$	$S(p)$	$S^t(p)$	$S(p)$
1	1.00	1.00	1.00	1.00
2	1.99	1.88	2.00	1.89
4	3.93	3.55	3.97	3.67
8	7.67	6.38	7.75	6.63
16	14.68	11.80	14.47	12.20

7. Conclusion

This paper focused on a parallel exact solution for TDNDP as a computing-intensive problem in transportation literature. Based on LeBlanc's B&B algorithm, we discussed that a synchronized MS parallelization paradigm adapted well to the solution algorithm due to processors' negligible communication and little idling overhead. The parallel B&B algorithm was developed and implemented in Java with search strategies of BFS and DFS. In a TDNDP instance with 12 projects, parallelization results over up to 16 processing cores in the Chicago Sketch network supported that:

- Due to traversing through B&B tree in the minimal computational effort, the parallel B&B algorithm with BFS strategy attained a notable faster performance than that of DFS. Consequently, the run-time for BFS strategy when using 16 processing cores was 17.06 minutes while it was 23.02 minutes for the strategy DFS;
- Both parallel strategies of BFS and DFS achieved an almost linear speedup, which slightly dropped by adding more processing cores. When using 16 processing cores, the achieved speedup values were

12.20 and 11.80 for the BFS and DFS strategies, respectively;

- The proposed algorithms were not subjected to an anomalous behavior of parallel B&B algorithms;
- The required memory for the strategy DFS increased almost linearly and proportional to the number of processing cores. The BFS strategy, on the other hand, required much more memory. However, this memory remained almost consistent as the number of cores changed.

8. Future work

To further extend the results of this paper, it is interesting to run the parallel B&B algorithms on massive parallelization facilities. In that case, application of other parallelization paradigms (e.g., asynchronous paradigms) may further reduce the idling and communication overheads and result in a more scalable parallel design. Sensitivity analysis over budget, demand level, network, and size of the projects is also a potential direction for future research. It would also be interesting to apply novel efficient TAP features, which can allow for enhancing the run-time as well as addressing low-capacity projects in the problem.

Acknowledgment

Authors wish to thank Prof. Hossain Poorzahedy for providing valuable comments during the research as well as Morteza Parvizi Omran and Saied Zarrinmehr for their generous cooperation. The IPM Grid Computing Group is sincerely acknowledged for providing facilities to run parallel programs. Authors also gratefully thank two anonymous referees and the editorial office of Scientia Iranica for providing constructive comments on this paper.

References

1. Poorzahedy, H. and Abulghasemi, F. "Application of ant system to network design problem", *Transportation*, **32**(3), pp. 251-273 (2005).
2. LeBlanc, L.J. "An algorithm for the discrete network design problem", *Transport. Sci.*, **9**(3), pp. 183-199 (1975).
3. Ben-Ayed, O., Boyce, D.E. and Blair, C.E. "A general bilevel linear programming formulation of the network design problem", *Transport. Res. B-Meth.*, **22**(4), pp. 311-318 (1988).
4. Poorzahedy, H. and Turnquist, M.A. "Approximate algorithms for the discrete network design problem", *Transport. Res.*, **16B**, pp. 45-55 (1982).
5. Farahani, R.Z., Miandoabchi, E., Szeto, W.Y. and Rashidi, H. "A review of urban transportation network design problems", *Eur. J. Oper. Res.*, **229**(2), pp. 281-302 (2013).
6. Roucairol, C. "Parallel processing for difficult combinatorial optimization problems", *Eur. J. Oper. Res.*, **92**(3), pp. 573-590 (1996).
7. Dantzig, G.B., Harvey, R.P., Lansdowne, Z.F., Robinson, D.W. and Maier, S.F. "Formulating and solving the network design problem by decomposition", *Transport. Res. B-Meth.*, **13**(1), pp. 5-17 (1979).
8. Solanki, R.S., Gorti, J.K. and Southworth, F. "Using decomposition in large-scale highway network design with a quasi-optimization heuristic", *Transport. Res. B-Meth.*, **32**(2), pp. 127-140 (1998).
9. Crainic, T.G., Le Cun, B. and Roucairol, C. "Parallel branch-and-bound algorithms", *In Parallel Comb. Optim.*, John Wiley & Sons, pp. 1-28 (2006).
10. Pekny, J.F. and Miller, D.L. "A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems", *Math. Program.*, **55**(1-3), pp. 17-33 (1992).
11. Anstreicher, K., Brixius, N., Goux, J.P. and Linderoth, J.T. "Solving large quadratic assignment problems on computational grids", *Math. Program.*, **91**(3), pp. 563-588 (2002).
12. Agrawal, J. and Mathew, T.V. "Transit route network design using parallel genetic algorithm", *J. Comput. Civil Eng.*, **18**(3), pp. 248-256 (2004).
13. Yang, Z., Yu, B. and Cheng, C. "A parallel ant colony algorithm for bus network optimization", *J. Comput. Aided Civil Infrastruct. Eng.*, **22**(1), pp. 44-55 (2007).
14. Cooper, I.M., John, M.P., Lewis, R., Mumford, C.L. and Olden, A. "Optimising large scale public transport network design problems using mixed-mode parallel multi-objective evolutionary algorithms", *IEEE Congress on Evolut. Comput. (CEC)*, Beijing, China, pp. 2841-2848 (2014).
15. Zarrinmehr, A. "Discrete network design using parallel branch-and-bound algorithms", M.Sc. Thesis, Dept. of Civil Eng., Sharif University of Technol., Tehran, Iran (2011).
16. Zarrinmehr, A. and Shafahi, Y. "Accelerating the performance of parallel depth-first-search branch-and-bound algorithm in transportation network design problem", *Proceedings of the Int. Conf. on Oper. Res. and Enterprise Sys. (ICORES)*, pp. 359-366 (2015).
17. Sheffi, Y., *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*, In Prentice Hall, Englewood Cliffs, NY, USA (1985).
18. Gendron, B. and Crainic, T.G. "Parallel branch-and-branch algorithms: Survey and synthesis", *Oper. Res.*, **42**(6), pp. 1042-1066 (1994).
19. Barney, B., *Introduction to Parallel Computing*, In Lawrence Livermore National Library (2010).
20. Lai, T.H. and Sahni, S. "Anomalies in parallel branch-

and-bound algorithms", *Commun. ACM*, **27**(6), pp. 594-602 (1984).

21. Pruul, E., Nemhauser, G. and Rushmeier, R. "Branch-and-bound and parallel computation: A historical note", *Oper. Res. Lett.*, **7**(2), pp. 65-69 (1988).

22. Li, G.J. and Wah, B.W. "Coping with anomalies in parallel branch-and-bound algorithms", *IEEE T. Comput.*, **100**(6), pp. 568-573 (1986).

23. Li, G.J. and Wah, B.W. "Computational efficiency of parallel combinatorial or-tree searches", *IEEE T. Software Eng.*, **16**(1), pp. 13-31 (1990).

24. Bar-Gera H. "Transportation network test problems", http://www.bgu.ac.il/ bargera/tntp (2015).

25. Boyce, D., Ralevic-Dekic, B. and Bar-Gera, H. "Convergence of traffic assignments: How much is enough?", *J. Transp. Eng.*, **130**(1), pp. 49-55 (2004).

Biographies

Amirali Zarrinmehr started his undergraduate studies in Computer Engineering and continued his BS in Civil Engineering at University of Tehran, Tehran, Iran. He earned an MS degree in Transportation Planning from Sharif University of Technology, Tehran, Iran. He is currently a PhD candidate of Highway and Transport Engineering at Tarbiat Modares University of Tehran, Tehran, Iran. His main research interests are design and application of combinatorial algorithms arising in transportation discipline. He is currently visiting the Northwestern University, IL, USA, as a pre-doctoral sabbatical scholar.

Yousef Shafahi received his BS from Shiraz University, Shiraz, Iran, MS from Isfahan University of Technology, Isfahan, Iran, and PhD from University of Maryland, College Park, MD, USA. He joined the Department of Civil Engineering at Sharif University of Technology, Tehran, Iran, as an Assistant Professor in 1998 and is now professor and head of transportation division in this department. His research interests are: urban transportation planning; road, rail, and air transportation; transit networks design and scheduling; applications of operation research; soft computing; and simulation in transportation.