

Methodology for generating extreme traffic demand patterns with optimization techniques

Conference Paper

Author(s):

Roca-Riu, Mireia; Ge, Qiao; Menendez, Monica

Publication date:

2015-04

Permanent link:

<https://doi.org/https://doi.org/10.3929/ethz-b-000100947>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Photo or figure (optional)

Methodology for generating extreme traffic demand patterns with optimization techniques

Mireia Roca-Riu, ETH Zürich
Qiao Ge, ETH Zürich
Mónica Menéndez, ETH Zürich

Conference paper STRC 2015

STRC

15th Swiss Transport Research Conference
Monte Verità / Ascona, April 15-17, 2015

Methodology for generating extreme traffic demand patterns with optimization techniques

Mireia Roca Riu
Institute for Transport
Planning and Systems ETH
Zurich, 8093, Zurich

Qiao Ge
Institute for Transport
Planning and Systems ETH
Zurich, 8093, Zurich

Mónica Menéndez
Institute for Transport
Planning and Systems ETH
Zurich, 8093, Zurich

Phone: +41766361988
Fax: +41 44 633 1057
email: mrocariu@gmail.com

Phone: +41 44 633 3249
Fax: +41 44 633 1057
email:qiao.ge@ivt.baug.ethz.ch

Phone: +41 44 633 6695
Fax: +41 44 633 1057
email:monica.menendez@ivt.baug.ethz.ch

April 2015

Abstract

Traffic scenario generation can aid analyzing and evaluating the performance of transportation systems under different traffic conditions. Unfortunately, it has not drawn much attention in practical applications. No standard scenario generation approach can be found so far, other than manual generation, random generation or exhaustive generation, which are neither accurate nor efficient.

Given its importance, we propose to borrow ideas from optimization techniques to solve the optimization problems in generating representative and/or extreme traffic scenarios. In this paper, five optimization methods, i.e., Linear Programming (LP), Dynamic Programming (DP), Greedy Algorithm (GR), Genetic Algorithm (GA), and Tabu Search (TS), are introduced and implemented to generate extreme traffic demand patterns.

The accuracy and efficiency of these methods are explored with a case study, in which the best and worst traffic demand patterns are searched for an abstract grid network. It is found in the case study that when the problem uses an independent traffic assignment model (e.g., shortest path assignment), LP and DP are the most accurate methods. GR is the most efficient method. GA presents certain accuracy when searching for best demand patterns, while TS is more accurate when searching for worst demand patterns. Moreover, GR, GA, and TS do not require linear constraints and objective functions, thus they could also work with other dependent traffic assignment models if computational times become affordable. Considering the overall performance (accuracy, efficiency, and constrains in application), GR is recommended as the best methods for generating extreme demand patterns.

Keywords

Demand pattern generation, Demand Scenario, Traffic Assignment, Traffic Networks

1. Introduction

Traffic scenario generation is an important tool for analyzing transportation systems. Through generating different traffic scenarios, relevant information can be gathered and used in short-term traffic forecasting, and/or to support the traffic control systems. Furthermore, for any specific traffic network, the traffic scenario generation can be employed for evaluating the performance of the network under different traffic conditions (e.g., specific events, different traffic demand patterns, and signal plans). This could help the transportation planner to evaluate the performance of the network design, and try to further improve it.

Despite its importance, however, to the best of the authors' knowledge, this topic has not drawn much attention in practical applications. In many cases, the traffic scenarios are generated manually by practitioners based on their assumptions (e.g., Scott et al., 2006). As one can imagine, the number of manually generated traffic scenarios is usually quite limited. Hence, planners and engineers are usually not able to thoroughly investigate all possible scenarios, and may omit certain important cases. On the other hand, using a brute force approach to produce all possible scenarios can be very time consuming for both the scenario generation process and the subsequent analysis of scenarios. Moreover, as some scenarios generated by an exhaustive search may only present minor differences between each other, the computation resources could be wasted in the assessment of those very similar scenarios. Therefore, the research of efficient methods that are able to generate a limited number of representative and/or extreme scenarios could be very valuable.

In several scientific research fields, optimization techniques are commonly used to find the best solution from many possible solutions. Since the problem of generating extreme traffic scenarios can be regarded as an optimization problem (i.e., finding the best or worst scenario from all possible scenarios), the ideas from these existing optimization techniques can be borrowed to efficiently solve the described problem.

This paper presents our findings in applying optimization methods to generate extreme traffic scenarios. It covers generating traffic demand patterns for a grid network that yield the best and worst traffic performance. The methods include linear programming, dynamic programming, greedy algorithm, genetic algorithm, and tabu search. The accuracy and efficiency of these methods are compared with a case study.

The paper is organized as follows: a brief review of the relevant optimization techniques is given first. Following the review, the problem is presented and the selected optimization methods for generating extreme traffic demand patterns are introduced in detail. Afterwards, a

case study is presented to illustrate and compare the performance of the different optimization methods. Conclusions and some recommendations for future research are given at last.

2. Brief review of optimization methods

This section provides a brief review of the optimization techniques. Note that these are general approaches, and there are a number of more specific optimization methods that fall within each category. The focus here lies on the methods that are most relevant to this study.

2.1 Mathematical Programming

Mathematical Programming (MP) represents a group of mathematical methods that are used to identify the “best” solution from a set of possible alternatives. There are many methods and applications available in this category, e.g., linear programming, integer programming, nonlinear programming, and dynamic programming. In this study, linear programming and dynamic programming are used.

Linear Programming

Linear Programming (LP) is a special form of MP. All constraints and objective functions in LP are in the linear form. The LP problem becomes an Integer Linear Programming (ILP) problem if all parameters are integers, or a Mixed Integer Linear Programming (MILP) problem if only some of the variables are integers. In contrast to the LP problem which can usually be efficiently solved, most ILP and MILP problems are NP-hard (i.e., the problem is at least as hard as any problem that is solvable in polynomial time, for details see Du and Ko, 2011).

The optimization problem in this study is a MILP problem. As the solution space of the MILP is non-convex, a systematic and potentially exhaustive search is needed. One classic method is Branch and Bound (Land and Doig, 1960). CPLEX (IBM ILOG, 2009), which is a powerful commercial solver that supports solving MILP with Branch and Bound approach among others, is employed in this study to solve the MILP problem. This method is able to provide the optimal solution of the problem, which allows to assess the results of other non-optimal procedures.

Dynamic Programming

Dynamic Programming (DP) was originally implemented in (Bellman, 1954). It solves a complex problem by first dividing the original problem into multiple sub-problems. When many sub-problems are the same or similar, DP will solve each individual sub-problem only once, and the solution will be stored for use if the same sub-problem is met next time. Thus, it can greatly save computational cost in searching the optimal solution when the problem has

many similar sub-problems. Finally, the solution of the original problem is obtained by combining the solutions of the sub-problems.

One major concern in DP is the dimensionality of the problem. Since DP is indeed an exhaustive search but without re-computing the solutions for the same sub-problems, the computational complexity of DP could increase exponentially along with the growth of dimensionality. This feature sometimes makes it infeasible for large-scale applications.

2.2 Heuristic Methods

When an exact method is not feasible due to extremely high computational cost, the heuristic methods can be applied to find an approximate solution. The heuristic methods are generally efficient, however, a truly optimal solution is not always guaranteed. According to (Marti and Reinelt, 2011), some of the important heuristic methods are decomposition methods, inductive methods, reduction methods, constructive methods, and local search methods. In this study, we employ a greedy algorithm, which belongs to the constructive methods, to solve the optimization problem.

Greedy Algorithm

Greedy Algorithm (GR) first decomposes the original problem into several stages (Cormen et al., 2001). Then it tries to find the locally optimal solution starting from the first stage, and the solution of any subsequent stage is based on the solution(s) of the previous stage(s). In general, a new decision made by GR only depends on the existing decisions but not on the future decisions. GR does not necessarily find the global optimal solution, however, it may approximate the global optimal with limited time requirements.

2.3 Metaheuristic Methods

The metaheuristic methods are higher-level heuristic methods with more intelligence. They consider the interaction between local improvement and higher level strategies, so that the searching process can be efficiently guided and the local optima could be avoided (Glover, 2003). They can provide satisfactory solutions with less computational effort than those simple heuristic methods (Blum and Roli, 2003).

Moreover, the metaheuristic methods usually make few assumptions of the optimization problem, thus they are independent of the specific contents of the problem being solved. This feature has made the metaheuristic methods quite attractive for solving a variety of

optimization problems, especially when there is little or no information available for the objective function (i.e., black-box function).

The most popular metaheuristic methods are genetic algorithm (Holland, 1973), simulated annealing (Kirkpatrick et al., 1983), scatter search (Glover, 1977), and tabu search (Glover, 1986). In this paper genetic algorithm and tabu search are used.

Genetic Algorithm

Genetic algorithm (GA), which was first developed in (Holland, 1973), mimics the natural selection process in solving a complex optimization problem. It first generates a random set (i.e., population) of possible solutions (i.e., phenotypes). Each phenotype is encoded by a genotype, normally in the binary form of 0s and 1s. Through calculating the fitness of all genotypes, some genotypes that best satisfy the fitness function are picked as parents. Then new offsprings are produced by crossing over the parents. To avoid getting the search stuck in local genotype space, mutations are introduced to alter the new offsprings. The old population is then replaced by the new offsprings. This is an iterative process (see Eiben and Smith, 2003 for more details), and it will be stopped when the maximum number of iterations (i.e., generations) is reached, and/or the solutions in the last few generations reach convergence (i.e., there is no improvement among these solutions).

One concern for GA is that it may converge toward local optima rather than global optimum, and the converging speed is greatly influenced by the initial population. In such case, one possible improvement is using different fitness functions, increasing the mutation rate, or employing other selection algorithms to retain certain diversity in the population (Taherdangkoo et al., 2013).

Tabu Search

Tabu Search (TS) is a local metaheuristic search approach introduced in (Glover, 1986). It is applied in several fields (Gendreau and Potvin, 2005). At each iteration step, some modifications of the solution based on the neighborhood are analyzed, and the best modification is chosen. To avoid local optima, TS accepts also the solutions that may not improve the objective function.

Before starting a new iteration, tabu status, long-term memory, and solution unfeasibility are checked. The tabu status is designed to prevent cycling but can be overridden if the solution improves the current best objective value (i.e., the aspiration criterion). Long-term memory and unfeasibility are techniques often employed to intensify or diversify the search process. Long-term memory stores some attributes of the solutions explored in order to guide the

search. In some cases, unfeasible solutions are accepted during some iterations in order to explore different search areas.

3. Methodology

3.1 Problem Description and Assumptions

In general, demand describes where vehicles enter the network and to which place they travel. It is defined in terms of quantities, as well as origins and destinations represented by trip matrices. Demand scenario generation is useful, as it can be employed for evaluating the performance of the network under different traffic conditions (e.g., specific events, different traffic demand patterns, signal plans). In particular, the generation of extreme demand patterns is interesting because it provides a range of performance of the given network. Our objective is to build the extreme patterns: the best and worst demand patterns for a given network.

The generation of demand patterns is associated with traffic assignment problem. In order to evaluate the goodness of a given scenario, we need to perform a traffic assignment to obtain performance indicators. Traffic assignment is the identification of the links and routes vehicles will use to cover their trip trying to emulate the real behavior of drivers. There are different algorithms to solve traffic assignment problem with different assumptions adapted to each context and problem. The shortest path algorithm (Dijkstra, 1959) is simple and easy to implement, however it is not very reliable when volumes and travel times increase. Algorithms based on User Equilibrium (UE) are still simple and have low memory requirements if solved under static conditions (Frank and Wolfe, 1956), even though computational effort is higher than shortest path. More realistic features are presented with dynamic models (Pedersen, 2011), but the complexity increases as well as the computing capacity needs. An algorithm for generating extreme demand pattern must include a balance approach representing a good compromise between efficiency and accuracy in them of traffic assignment.

Our long-term proposal is to build an algorithm divided in two consecutive phases: the demand pattern generation and the demand pattern evaluation. In the first phase, shortest path assignment will be used. Although it is not very accurate, it is rather efficient and allows the algorithm to run the traffic assignment many times with a low computational cost. In the second phase (evaluation), the best and worst demand patterns found in the first phase will be evaluated, this time with a higher quality traffic assignment.

The focus of this paper is only the first phase of the algorithm, the generation phase. The generation problem can be formulated as an optimization problem. The objective is to propose a demand pattern that minimizes or maximizes the traffic performance in a given network. We

investigate the accuracy and efficiency of multiple optimization methods for the generating extreme demand patterns.

We employ a simple n -by- n grid network in this study, although the methodology could be applied to any network with different topologies. In addition, the following assumptions are made to simplify the problem:

- 1) All trips are generated and ended in the middle of the link, i.e., the demand nodes.
- 2) The traffic is uniformly distributed in terms of time.
- 3) The trips between any two demand nodes in the network can only be either T or 0 (i.e., no trips).
- 4) As stated previously the traffic assignment used in this phase of the algorithm will be the Shortest Path Algorithm. That means, traffic assignment from one origin-destination (OD) pair is independent of the traffic assignment from any other OD pair. This assumption is not very realistic in traffic, but it gives an approximation of the performance in a very simple way and with very few computational efforts. In particular, the simplicity allows LP and DP methods to be applied. These methods might be able to obtain the exact solutions. GR, GA, and TS could work with other traffic assignment methods, but the computational requirements are still too high to obtain results with reasonable computational times. In any case, all methods will be compared with shortest traffic assignment.

In an n -by- n grid network, there are $2n(n-1)$ links and $2n(n-1)$ demand nodes (1st assumption). We consider demand node o and demand node d ($o, d \in [1, 2n(n-1)], d \neq o$) as the origin and destination node, respectively. Thus, there are a total of $N^* = 4n^4 - 8n^3 + 2n^2 + 2n$ possible OD pairs.

Given the 3rd assumption, for a total demand of $N \times T$ ($N \in [1, N^*]$), the corresponding demand pattern (i.e., a certain combination of OD pairs) should therefore meet the following requirement: N OD pairs with T trips each, and the other $N^* - N$ OD pairs with no trips. The total number of possible demand patterns that satisfy such condition is $N^*! / [N! \times (N^* - N)!]$. Accordingly, the problem of generating the extreme traffic demand patterns can be considered as a combinatorial optimization problem, i.e., the target is to find out one or several demand patterns that yield the best or worst traffic performance from the $N^*! / [N! \times (N^* - N)!]$ demand patterns.

Due to the extremely large number of combinations, an exhaustive search or a totally randomized search could be an unfeasible and tedious process. To solve the combinatorial

optimization problem with both accuracy and efficiency, we borrow some ideas from well-known optimization techniques, and implement them below.

Moreover, we employ the maximum flow across all links in the network as the traffic performance indicator. The worst demand pattern should maximize the maximum flow across all links, while the best demand pattern should minimize the maximum flow across all links. Note that other indicators such as maximum link travel time can also be used with these algorithms. However, according to the BPR function (Bureau of Public Roads, 1964), the travel time monotonically increases along with the growth of traffic flow when the road has a fixed capacity. Therefore, it is expected that the worst or best demand patterns found using the maximum flow across all links are the same as those found using the maximum link travel time.

3.2 Linear Programming

A MILP model is proposed for finding both best and worst demand patterns. A binary vector $X = [x_1, x_2, \dots, x_{N^*}]$ is used to define a possible demand pattern. x_i ($i \in [1, N^*]$) is 1 when OD pair i has T trips, otherwise it is 0. Then we define function t_a as the traffic assignment function. It calculates $f^k(X)$, the total flow on a specific link k by a given demand pattern X :

$$f^k(X) = t_a(X, k) \quad k \in [1, 2n(n-1)] \quad (1)$$

Given the 3rd assumption, the value of $f^k(X)$ will be a multiple of T or 0. Since the traffic assignments from different OD pairs are independent (i.e., the 4th assumption), the function t_a is an additive linear function:

$$t_a(X, k) = \sum_{i=1}^{N^*} t_a(X_i, k) * x_i \quad (2)$$

where X_i means all elements in X are 0s, except the i -th element x_i that is 1. $t_a(X_i, k)$ calculates \bar{f}_i^k , i.e., the flow on link k if only the OD pair i has T trips.

$$\bar{f}_i^k = t_a(X_i, k) \quad (3)$$

Due to the 4th assumption, \bar{f}_i^k ($i \in [1, N^*], k \in [1, 2n(n-1)]$) can be pre-computed and directly used in the subsequent computation. Moreover, because of the 3rd assumption, \bar{f}_i^k will be either T or 0. Then Equation (1) is simplified by combining Equations (2) and (3):

$$f^k(X) = \sum_{i=1}^{N^*} \bar{f}_i^k * x_i \quad (4)$$

We use variable z to indicate the maximum flow across all links in the network. When searching for the best demand pattern, the objective is to minimize z . The formulation of the best demand pattern with respect to any demand pattern X must satisfy the following constraints:

$$\sum_{i=1}^{N^*} x_i = N \quad (5)$$

$$f^k(X) = \sum_{i=1}^{N^*} \bar{f}_i^k * x_i \leq z \quad \forall k \in [1, 2n(n-1)] \quad (6)$$

Constraint (5) limits the number of OD pairs with T trips to N . Constraint (6) imposes that variable z has to be not less than any $f^k(X)$.

When searching for the worst demand pattern, the objective is to maximize z , but with only the above constraints the problem would be unbounded. To solve this problem, we introduce an extra binary variable ζ_k ($k \in [1, 2n(n-1)]$) that takes the following values:

$$\zeta_k = \begin{cases} 1 & \text{if } \sum_{i=1}^{N^*} \bar{f}_i^k * x_i = z \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in [1, 2n(n-1)] \quad (7)$$

The following constraints are used to translate Equation (7) into a linear form:

$$\sum_{k=1}^{2n(n-1)} \zeta_k \geq 1 \quad (8)$$

$$\sum_{i=1}^{N^*} \bar{f}_i^k * x_i + M(1 - \zeta_k) \geq z \quad \forall k \in [1, 2n(n-1)] \quad (9)$$

Constraint (8) guarantees that at least one link will have the maximum flow. Constraint (9) is active (i.e., equality holds) only when $\zeta_k = 1$. When link k has the maximum flow, then variable $\zeta_k = 1$, and Constraint (9) gives the upper bound of variable z . We set M to a sufficiently large value (e.g., $M = N \times T$), so that Constraint (9) does not affect the variables (i.e., inequality holds) when $\zeta_k = 0$. The above MILP model is implemented in Optimization Programming Language (OPL) and solved with CPLEX 12.1.

3.3 Dynamic Programming

The idea of DP is dividing a complex optimization problem into several sub-problems, and combining the solutions of all sub-problems in the end to form the final optimal solution. In this study, to find the worst demand pattern, the first step is to find out which demand patterns can maximize the flow on individual links.

Due to the 4th assumption that the traffic assignments from different OD pairs are independent, to maximize the flow on a given link k ($k \in [1, 2n(n-1)]$) is analogically a “0-1 knapsack problem” (for details see Martello et al. 1999). Assume that the capacity of a knapsack is N (i.e., the total demand is $N \times T$, $N \in [1, N^*]$), and there are in total N^* items (i.e., N^* OD pairs). The weight of each item is 1, and its value is either T or 0. The original problem (i.e., maximizing the flow on link k) is the same problem as finding out which items should be put into the knapsack, so that the total value of all items in the knapsack is maximized. Below is the pseudo code for computing the maximum flow on link k , under the constraint that only N OD pairs have T trips:

```

for w=1 to N+1
   $h_{DP}(1,w)=0$ 
end
for i=1 to  $N^*+1$ 
   $h_{DP}(i,1)=0$ 
end
for i = 1 to  $N^*$ 
  for w = 1 to  $N$ 
     $h_{DP}(i+1,w+1) = \max[h_{DP}(i,w+1), \bar{f}_i^k + h_{DP}(i,w)]$ 
  end
end

```

h_{DP} is a (N^*+1) -by- $(N+1)$ matrix storing the maximum flow by different demand patterns, and \bar{f}_i^k is the traffic flow on link k if only OD pair i has T trips (Equation (3)). The maximum flow on link k is therefore $h_{DP}(N^*+1, N+1)$. Through backtracking, we can determine which OD pairs have T trips in the worst demand pattern.

If the above searching process is applied to all links, then we can find the worst demand pattern(s) for each individual link, namely, $W_N^1, \dots, W_N^{2n(n-1)}$. The corresponding maximum flows on individual links are respectively $mf_1, mf_2, \dots, mf_{2n(n-1)}$. In the next step, we cross compare these maximum flows on individual links. Suppose mf_{k^*} ($k^* \in [1, 2n(n-1)]$) is larger than other flows, then $W_N^{k^*}$ is the worst demand pattern.

Unfortunately, the above method does not work for the best demand pattern case. The reason is the demand pattern that minimizes the maximum flow on one individual link will not necessarily minimize the maximum flow in the network. Thus, in this paper we only use DP to compute the worst demand pattern.

3.4 Greedy Algorithm

The greedy algorithm is based on the method described in (Ge and Menendez, 2014). It implements the same searching process for the best and worst demand patterns except the

target functions. For the ease of illustration, here we just show the searching process for the worst demand pattern, in which N ($N \in [1, N^*]$) OD pairs have T trips while all other OD pairs have no trips.

We use the same binary vector in LP, i.e., $X = [x_1, x_2, \dots, x_{N^*}]$, to represent a possible demand pattern. Using Equation (4), the maximum flow across all links in this network with respect to demand pattern X (i.e., $h_{GR}(X)$) can be defined as:

$$h_{GR}(X) = \max_{k \in [1, 2n(n-1)]} \{f^k(X)\} = \max_{k \in [1, 2n(n-1)]} \left\{ \sum_{i=1}^{N^*} \bar{f}_i^k * x_i \right\} \quad (10)$$

In the first step, we produce a set of N^* vectors, namely, X_{p_1} ($p_1 \in [1, N^*]$). X_{p_1} is a binary vector with N^* elements: all elements are 1s except the p_1 -th element that is 0. Accordingly, the maximum flow across all links are $h_{GR}(X_{p_1})$ ($p_1 \in [1, N^*]$). Suppose $h_{GR}(X_{w_1})$ ($w_1 \in [1, N^*]$) is larger than all other maximum flows, then vector X_{w_1} is the optimal solution in this step.

In the second step, we generate a set of N^*-1 vectors based on X_{w_1} , namely, X_{w_1, p_2} ($p_2 \in [1, N^*]$, $p_2 \neq w_1$). All elements in X_{w_1, p_2} are 1s, except the w_1 -th and p_2 -th elements that are 0s. Again, we can evaluate the corresponding maximum flow across all links $h_{GR}(X_{w_1, p_2})$. The vector X_{w_1, w_2} ($w_2 \in [1, N^*]$, $w_2 \neq w_1$) is selected as the solution in the current step if $h_{GR}(X_{w_1, w_2})$ is larger than all other maximum flows.

In the next step, we do the same but based on X_{w_1, w_2} , etc. In each step, we change one non-zero element to 0 based on the solution from the previous step. The iteration will stop when the vector $X_{w_1, w_2, \dots, w_{N^*-N}}$ ($w_1, \dots, w_{N^*-N} \in [1, N^*]$, $w_1 \neq w_2 \neq \dots \neq w_{N^*-N}$) is found. The worst demand pattern found by the greedy algorithm is the following: OD pairs w_1, \dots, w_{N^*-N} have no trips, while all other N OD pairs have T trips.

3.5 Genetic Algorithm

GA starts with generating several random solutions (i.e., phenotypes). These solutions are encoded in binary strings (i.e., genotype) with 1s and 0s: if OD pair i has T trips in a certain demand pattern, the i -th gene in the genotype is 1, otherwise it is 0. As a rule-of-thumb, to completely scan the solution space, the population size P should be at least twice the number of parameters. However, a large size will certainly increase the computational cost. Therefore, in this study, considering both accuracy and efficiency, we manually set $P = 2N^*$.

Suppose that in a certain generation, x_i^p ($i \in [1, N^*]$, $p \in [1, P]$) is the i -th gene in the p -th genotype X^p (i.e., $X^p = [x_1^p, x_2^p, \dots, x_{N^*}^p]$). As there are only N OD pairs with T trips in one demand pattern, the following constrain is used:

$$\sum_{i=1}^{N^*} x_i^p = N \quad \forall p \in [1, P] \quad (11)$$

Similar to Equation (4), in GA, the fitness function with respect to X^p is:

$$h_{GA}(X^p) = \begin{cases} \max_{k \in [1, 2n(n-1)]} \left\{ \sum_{i=1}^{N^*} x_i^p * \bar{f}_i^k \right\} & \text{if } \sum_{i=1}^{N^*} x_i^p = N \\ \text{penalty} & \text{otherwise} \end{cases} \quad (12)$$

where \bar{f}_i^k is defined according to Equation (3). The algorithm will minimize (maximize) the fitness function h_{GA} when searching for the best (worst) demand pattern. The parameter *penalty* is used when the genotype is not feasible (i.e., Constrain (11) is not satisfied). It is set to $N \times T$ when searching for the best demand pattern, or 0 when searching for the worst demand pattern.

Starting from the first generation, all randomly generated solutions will be evaluated using Equation (12). Then, the 1% of the population that best fit the fitness function will be selected as the parents. In this study, one-point cross over is employed in GA, i.e., two offsprings are created by exchanging the genes on the tails of the parents starting from a randomly picked point. The cross-over rate is 80%, i.e., when producing offsprings, 80% of the offsprings are produced by crossing over two randomly selected parents, while the other 20% are copies of the parents. In addition, we randomly mutate 10% of the offsprings. A gene will be randomly picked and inverted (i.e., if the original gene is 0, it is changed to 1, and vice versa) in mutated offsprings. The P offsprings are then used as the new population in the next generation by repeating the previous process.

The above iteration process will stop when there is no improvement in the fitness function for the last 20 generations (i.e., the solutions have reached convergence), or when a maximum of 200 generations is reached. The solution in the last iteration that best fits the fitness function is taken as the final solution.

In this study, GA is implemented with the Matlab Global Optimization Toolbox (Matlab, 2003).

3.6 Tabu Search

TS is a meta-heuristic that has been tested in several transport problems with successful results (Glover, 1990). The search begins with a randomly generated solution. At each step, several neighborhoods of the solution are tested. Then the search moves to a new solution, even though it might not improve the objective value.

The neighborhood of the solution is defined by different modifications of the solution. We use the states of all OD pairs to form a solution: the state of an OD pair is 1 when it has T trips, or 0 when it has no trips. The modification can be either changing the state of one OD pair from 1 to 0, or from 0 to 1, so that the objective value is improved at each step. To decide which OD pairs are modified in each step, we consider both random and non-random selections with probability of 20% and 80% respectively. The non-random selection utilizes information from existing solution. Suppose in a solution, link k is found with maximum flow across all links, to influence the value of the objective function, the modifications should be made on the OD pairs that are expected to have trips passing through this link. For example, when searching for the best demand pattern that minimizes the maximum flow, we change the state of an OD pair from 1 to 0 if the trips from this OD pair are most likely to pass through link k (e.g., when the OD pairs are near link k), and change the state of another OD pair from 0 to 1 if the trips from this OD pair are not likely to pass through link k .

Moreover, we use a tabu label to prevent the algorithm from getting stuck in the same solutions. When the state of an OD pair is changed from 1 to 0 in a new solution, this OD pair is labeled as tabu for a certain number of iterations. During these iterations, the state of this OD pair cannot be changed back to 1 again in the new solutions. Similarly, when the state of an OD pair is changed from 0 to 1 and it is labeled as tabu, the state of this OD pair cannot be changed back to 0 in the solutions for several iterations.

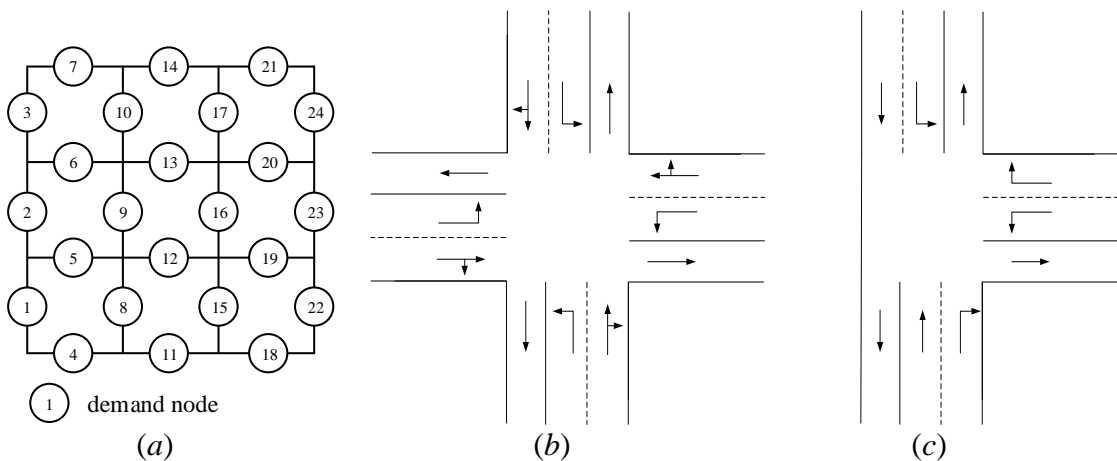
Note that the search may be dependent on the initial solution and converge to a local optimal solution. To avoid this effect, when it is not able to provide better solutions after several iterations, the algorithm is reinitialized. It will start either from the best available solution, or from another randomly generated solution.

4. Case Study and Results

4.1 Design of the Case Study

An abstract 4-by-4 grid network with bidirectional streets (Fig. 1a) is employed in the case study. Each link in the network is 100 m long. The reason for using this small network is for the ease of deriving the optimal best and optimal worst demand patterns, so that it is feasible to cross compare the performance of different optimization methods. Note that it is not necessary to use a small network in practice, and there is no specific constrain for the size of the network when applying the abovementioned optimization methods.

Figure 1 (a) Layout of the 4-by-4 grid network. (b) Layout of the 4-leg intersection. (c) Layout of the 3-leg intersection.



This network has two types of intersections:

- 4-leg intersections (Fig. 1b) inside the network. The right turn and the through traffic are grouped in one lane, while the other lane is dedicated to left turners.
- 3-leg intersections (Fig. 1c) in the periphery. The left turn, right turn, and the through traffic are all separated in different lanes.

As assumed in the previous section, all demand nodes are located in the middle of the link (Fig. 1a). Hence, there are 24 demand nodes, and $N^* = 552$ OD pairs. In a given demand pattern, the trips from any OD pair can only be T or 0 . We further assume that capacity of the link is infinite as congestion is not considered. Moreover, all vehicles have a constant speed of 50 km/h.

A shortest path algorithm is used for the traffic assignment. It applies the Dijkstra's algorithm (Dijkstra, 1959) to determine one single route with the shortest total travel distance from the origin node to the destination node. It assigns all trips of the corresponding OD pair to this

shortest route independent of the traffic assignments from other OD pairs, which is in accordance with the 4th assumption in the previous section. Different green times are given to the left turners (2.5s) and right turners (22.5s) at intersections. Hence, when two routes have the same distance, the traffic will be assigned to the route with less left turns. The shortest path based traffic assignment can be found in simulations such as (Ortigosa and Menendez, 2014) and (Gora, 2009). For the ease of comparing different optimization algorithms, it is employed here because it takes much less computation time than other complex assignment methods such as dynamic assignment.

The maximum flow across all links is used as the traffic performance indicator. The objective functions are hence minimizing the maximum flow when searching for the best demand patterns, and maximizing the maximum flow when searching for the worst demand patterns. The experiment will be performed with hourly total demands from T to $552T$ (i.e., 552 cases) for both best and worst demand patterns.

4.2 Results

The five optimization methods, Linear Programming (LP), Dynamic Programming (DP), Greedy Algorithm (GR), Genetic Algorithm (GA), and Tabu Search (TS), are used to generate the extreme traffic demands in the case study. The corresponding results, i.e., maximum flow across links for different total demands, are shown in Fig. 2.

Fig. 2a shows the results from LP and DP (note that DP is used only for finding the worst demand patterns). It shows that LP and DP yield exactly the same maximum flows for all cases. Since the traffic assignment model adopted here is an independent linear model (Equation (2)) and LP and DP are exact optimization methods, their results are the truly optimal solutions. We use them as reference in evaluating the accuracy of the other methods. Moreover, we randomly generate 50 demand patterns for all 552 cases, and the averages of the corresponding maximum flows are plotted in Fig. 2a. Obviously, in almost all cases the randomly generated demand patterns are neither best nor worst demand patterns. It further highlights the importance of this research, as the use of a randomly generated demand patterns could bring incorrect results (e.g., overestimating or underestimating the capacity of the network) in network design and evaluation.

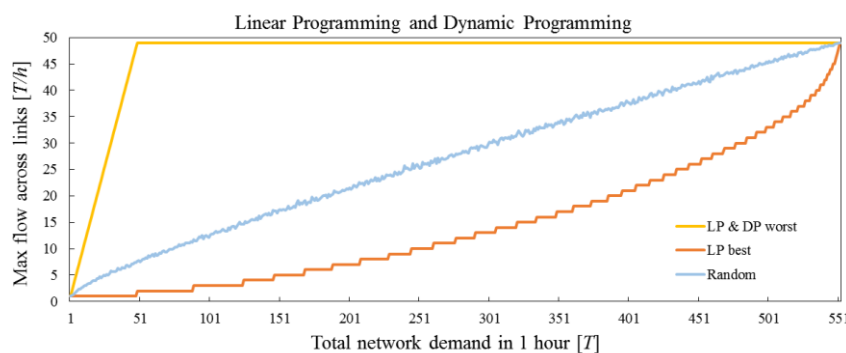
The results from GR, GA and TS are shown in Fig. 2b, 2c, and 2d, respectively. GR finds the optimal solutions in all worst demand pattern cases, but in best demand pattern cases some errors exist when the total demand is between $255T$ and $530T$. The results of GA are generally satisfactory when searching for the best demand patterns with some errors around $270T$ and $530T$; on the contrary, when searching for the worst demand patterns, significant errors exist

in most cases (from 48T to 450T). TS has the opposite behavior as GA: it performs well in most worst demand pattern cases, but it also shows some errors in a few cases (e.g., 101T); in the best demand pattern cases, however, significant errors exist in almost all cases except when the total demands are over 525T.

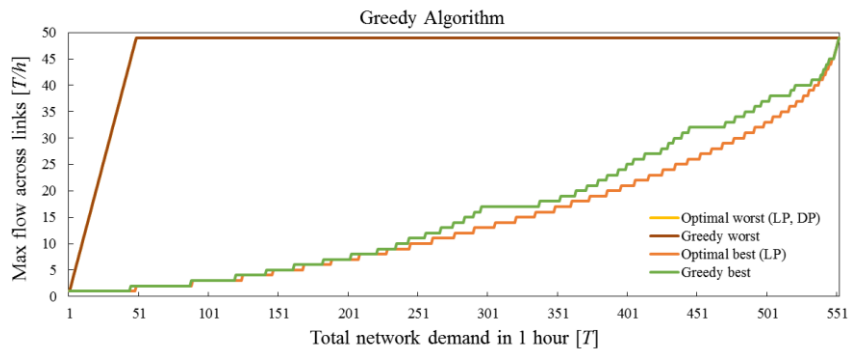
Fig. 3a and 3b cross-compare the accuracy of GR, GA and TS in terms of relative error. Note that LP and DP are not included since they are used as reference. The boxplot in Fig. 3a shows that when searching for the best demand patterns, GR and GA have less variations of relative errors ($\sigma^2_{GR} = 0.02$ and $\sigma^2_{GA} = 0.04$) than TS ($\sigma^2_{TS} = 0.34$). Moreover, 75% of the cases using GR and GA have relative errors under 0.3, while less than 25% of the TS cases can reach similar accuracy. In other words, GR and GA are generally more accurate than TS in generating the best demand patterns in the case study, with GR showing a slightly higher accuracy than GA.

The boxplot in Fig. 3b shows that when searching for the worst demand patterns, GR obviously outperforms GA and TS without any error. With TS, 75% of the cases have relative errors under 0.25, although there are few cases with extreme errors between 0.05 and 0.22. With GA, less than 25% of the cases have relative errors under 0.05, and the maximum relative error is 0.18. Therefore, GR is the most accurate method in generating the worst demand patterns in this case study, followed by TS, and GA in the last place.

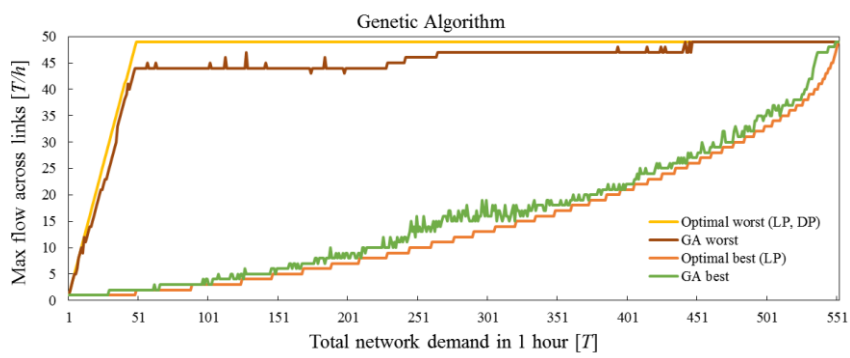
Figure 2 Maximum flows across links for different total demands in worst and best cases obtained via (a) linear programming and dynamic programming; (b) greedy algorithm; (c) genetic algorithm; (d) tabu search.



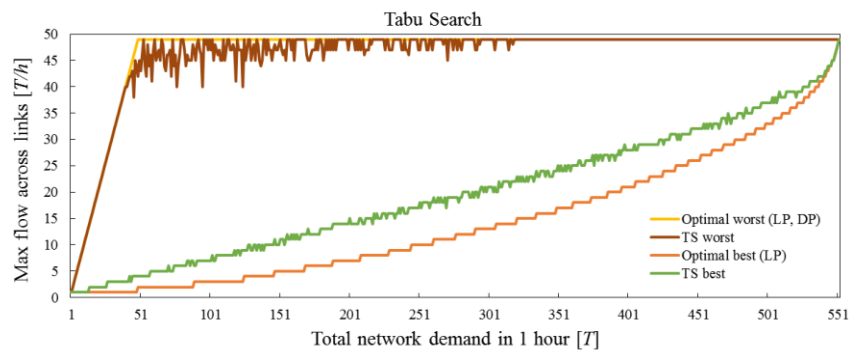
(a)



(b)

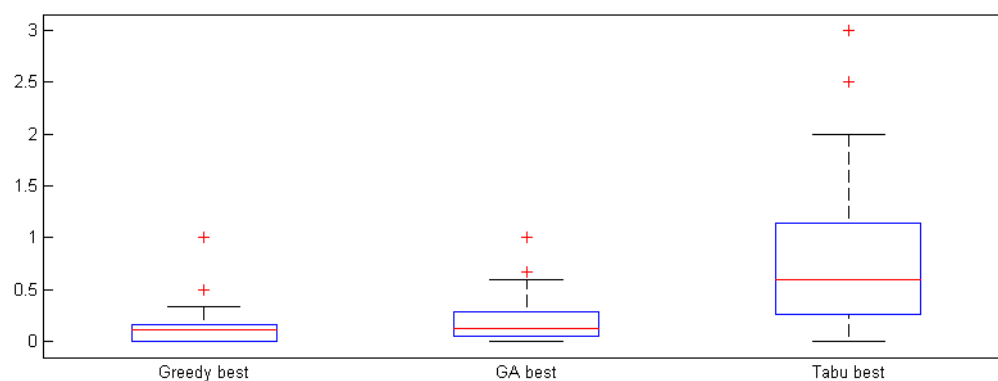


(c)

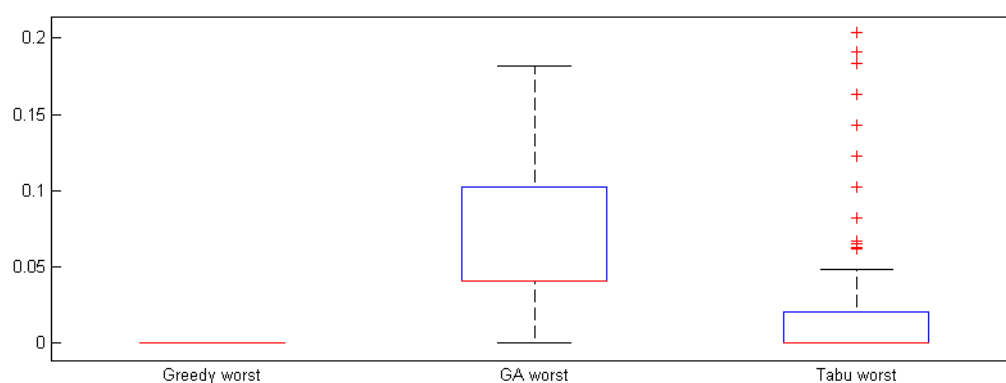


(d)

Figure 3 (a) Relative errors in finding the best demand patterns by GR, GA and TS; (b) Relative errors in finding the worst demand patterns by GR, GA and TS.



(a)



(b)

Table 1 compares the computational cost of the five methods based on average number of evaluations of the objective function per case, and the average running time per case. Obviously, GA and TS require the most computational cost, while GR requires the least computational cost. Hence, the ranks of the five methods in terms of efficiency are $GR > LP \approx DP \gg TS > GA$.

Table 1 Computational Cost of the Five Optimization Methods

Method	Average number of evaluations of the objective function per case	Average running time per case (s)
LP	Not relevant*	0.27
DP	13248	0.23
GR	277	0.13
GA	600000	900
TS	50000	216.2

*This is because CPLEX does not evaluate the objective function many times to optimize the problem.

5. Conclusions

In this paper, we introduce and implement five optimization methods, i.e., Linear Programming (LP), Dynamic Programming (DP), Greedy Algorithm (GR), Genetic Algorithm (GA), and Tabu Search (TS), to generate the extreme demand patterns that minimize / maximize the maximum flow across links in an abstract grid network. The following conclusions are made based on our findings in the case study.

In the demand pattern generation, the traffic assignment model used is the shortest path, given its simplicity and short computational time. Plus, that allows LP and DP to be the most accurate methods for generating extreme demand patterns. LP can find both the optimal best and optimal worst demand patterns, while DP only works for finding the optimal worst demand patterns. The main limitation of LP and DP is that they cannot be used with other traffic assignment models. Moreover, in the case of LP the solution method depends on the use of a commercial software. In any case, LP and DP provide in the test case the optimal results. This allows us to assess the goodness of the other methods.

GR is the only method besides the two exact methods LP and DP in this study that finds the optimal worst demand patterns in all cases. Although it has some errors in finding the best demand patterns, the overall accuracy is still acceptable. Moreover, GR is the most efficient method among the five methods, and it requires the least number of evaluations of the objective function. This makes it very attractive if other traffic assignment methods (e.g., Ge et al. 2014) are considered in the generation phase, although the global optima is not necessarily guaranteed. Also, the algorithm implementation is extremely easy.

The two metaheuristic methods GA and TS perform differently in the case study. GA has high accuracy in generating best demand patterns, but the accuracy in generating the worst demand patterns is not satisfactory. On the contrary, TS works relatively well for generating worst demand patterns, but it has significant errors when generating best demand patterns. Moreover, GA and TS generally require more computational cost than the other methods. However, since GA and TS do not require linear constraints and objective functions, they can also solve problems that adopt dependent traffic assignment models. When working with dependent traffic assignment model, GA and TS are expected to be more accurate than GR when a sufficiently big number of iterations are employed. However, this will certainly further increase the computational cost, and limit their usage with computationally cheap problems.

To sum up, based on the overall performance (i.e., accuracy, efficiency, and constraints in application), GR is the best method in this research so far. It is possible that GA and TS can perform better when computational cost is negligible, but if efficiency is taken into account, it

is more feasible to use GR to solve the problem. Future research will be devoted to develop the evaluation phase of the algorithm. The extreme demand patterns obtained in the first phase will be evaluated with more accurate traffic assignment methods. Also, the methodology will be applied to a bigger grid network.

Acknowledgements

The authors are very grateful to Mr. Javier Ortigosa from ETH Zurich, for providing us the grid network and traffic assignment model used in the case study.

6. References

- Bellman, R. (1954) The theory of dynamic programming, *Bulletin of the American Mathematical Society*, **60** (6) 503-516.
- Blum, C., and A. Roli (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Computing Surveys (CSUR)*, **35** (3), 268-308.
- Bureau of Public Roads (1964) Traffic assignment manual, Washington DC: US Department of Commerce, Urban Planning Division.
- Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001) *Introduction to algorithms*, **2**, Cambridge: MIT press.
- Dijkstra, E. W. (1959) A note on two problems in connexion with graphs, *Numerische Mathematik*, **1**, 269–271.
- Du, D. Z., and K. Ko (2011) *Theory of computational complexity*, **58**. John Wiley & Sons.
- Eiben, A. E., and Smith (2003) J. E. *Introduction to evolutionary computing*, Springer.
- Frank, M. and P. Wolfe (1956) An algorithm for quadratic programming, *Naval Research Logistics Quarterly*, **3** (1-2) 95-110.
- Ge, Q. and M. Menendez (2014) An Efficient Sensitivity Analysis Approach for Computationally Expensive Microscopic Traffic Simulation Models, *International Journal of Transportation*, **2** (2) 49-64.
- Ge, Q, J. Ortigosa, and M. Menendez (2014) Traffic demand pattern generation for a grid network based on experiment design. Paper presented at the *14th Swiss Transport Research Conference*, Ascona, May 2014, Switzerland.
- Gendreau, M. and J. Potvin (2005) *Tabu Search*. In *Search Methodologies*. Burke, E. and G. Kendall. Springer.
- Glover, F. (1977) Heuristics for Integer programming Using Surrogate Constraints, *Decision Sciences*, **8** (1) 156-166.
- Glover, F. (1986) Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, **13** (5) 533-549.
- Glover, F., and G. A. Kochenberger (2003) *Handbook of metaheuristics*. Springer.
- Glover, F. (1990) Tabu Search - Part II, *ORSA Journal on Computing*, **2** (1), Operation Research Society of America.
- Gora, P. (2009) Traffic Simulation Framework - a cellular automaton-based tool for simulating and investigating real city traffic. *Recent Advances in Intelligent Information Systems*, 642-653.
- Holland, J. H. (1973) Genetic algorithms and the optimal allocation of trials, *SIAM Journal on Computing*, **2** (2) 88-105.

-
- Kirkpatrick, S., C. D. Gelatt Jr., and M. P. Vecchi (1983) Optimization by Simulated Annealing, *Science*, **220** (4598) 671-680.
- IBM ILOG CPLEX V12.1: (2009) *User's Manual for CPLEX*, International Business Machines Corporation.
- Land A.H., and A.G. Doig (1960) An automatic method of solving discrete programming problems, *Econometrica*, **28** (3) 497-520.
- Martello, S., D. Pisinger, and P. Toth (1999) Dynamic programming and strong bounds for the 0-1 knapsack problem, *Management Science*, **5** (3) 414-424.
- Martí, R., and G. Reinelt (2011) *The linear ordering problem: exact and heuristic methods in combinatorial optimization*. Springer.
- Ortigosa, J., and M. Menendez (2014) Traffic performance on a quasi-grid urban structures, *Cities*, **36** 18-27.
- Pedersen, N.J. (2011) Dynamic Traffic Assignment. A primer, Transportation research circular Number E-C154. Transportation Research Board of the national academies.
- Scott, D., D. Novak, L. Aultman-Hall, and F. Guo (2006) Network robustness index: A new method for identifying critical links and evaluating the performance of transportation networks, *Journal of Transport Geography*, **14** (3) 215-227.
- Taherdangkoo, M., M. Paziresh, M. Yazdi, and M. H. Bagheri (2013) An efficient algorithm for function optimization: modified stem cells algorithm, *Central European Journal of Engineering*, **3** (1) 36-50.
- The MathWorks (2013), Inc. *MATLAB R2014a Global Optimization Toolbox - User's Guide*.