A Genetic Algorithm with Cycle Representation and Contraction Digraph Model for Guideway Network Design of Personal Rapid Transit

Jin-Myung Won and Fakhreddine Karray

Abstract—In this paper, we propose a steady-state genetic algorithm (GA) with cycle-based representation and a contraction digraph model to deal with the guideway network design problem of personal rapid transit (PRT). PRT is a novel transportation paradigm, where many computer-controlled vehicles running on an elevated guideway network (GN). A GN may contain hundreds of guideway links and how to design the minimum-cost feasible GN is a challenging problem. Given a set of stations, the proposed GA models a candidate GN as a union of one or more simple directed cycles visiting two or more stations. This cycle representation not only provides good solution locality but allows us to establish a contraction digraph model, where its feasibility can be efficiently evaluated. We also develop special genetic operators well suited for the cycle representation. Numerical experiments conducted for various problem instances show the proposed GA outperforms the conventional ones once the solution is represented by a moderate number of cycles.

I. INTRODUCTION

As passengers expect more on their daily transportation experiences, many researchers have proposed a novel type of transit systems, which provides high standard of safety and service quality. Personal rapid transit (PRT) is a good example of such a novel transportation concept [1], [2]. A PRT system comprises an elevated guideway network (GN) and many computer-controlled compact vehicles running over the GN. The GN interconnects many small stations, which are easily accessible from everywhere in a metropolitan area. With PRT, a passenger party enjoys an on-demand, private, non-stop, and non-transfer trip for their destination station.

PRT is a network-based transportation system and its cost and performance strongly depends on the GN topology. A GN should be designed so that it could minimize construction cost and worst-cast guideway link traffic. If the latter exceeds traffic capacity of the link, the system experiences traffic congestion, which drastically degrades passenger convenience and even causes system failure. Given station locations and the worst-case inter-station traffic demands, the objective of the GN design problem (GNDP) is to find the minimum cost guideway link set that keeps the worst-case link traffic no greater than the link capacity.

The network design problems like a GNDP are found in many engineering fields such as telecommunication, computer network, resource allocation, and transportation. The network design problem is NP-complete and an enumeration-based method fails to find a good solution in a reasonable computation time even with a moderate network size. To handle the GNDP, we can customize various heuristic methods such as greedy heuristics, branch and bound methods, tabu search techniques [3], simulated annealing algorithms [4], [5], and ant colony optimizers [6]. Nonetheless, a genetic algorithm (GA) is known as one of the best choices since it finds good solutions in an acceptable computation time regardless of the network size [7], [8], [9], [10], [11], [12], [13], [14].

When utilizing a GA, how to represent a candidate network is a critical question to determine the performance of the GA. The representation methods reported up to now can be classified into two categories: edge representation and node representation [8]. The edge representation employs a binary string, where each bit indicates the presence of an edge in the candidate network [9], [12]. If the super-network is complete, this binary string can be directly mapped to the node adjacency matrix. An inherent weak point of the edge representation is the standard genetic operators mostly generate infeasible or expensive networks, which should be repaired by other heuristics [9].

Typical node representation methods include Prüfer encoding [15] and determinant encoding [16], which were developed to model a spanning tree. The Prüfer encoding provides one-to-one correspondence between a spanning tree and the associated Prüfer string. However, the Prüfer encoding has a poor solution locality and a slight change in a Prüfer string may cause a drastic change in the tree. This poor solution locality is the main reason why it is difficult to apply the node representation to the GA [8]. Moreover, the spanning tree is poor in representing a one-way circulator-based network such as the GN.

To overcome these limitations of the existing representation methods for the GNDP, this paper proposes a steady-state GA with a new representation technique, which represents a candidate GN with a union of one or more simple cycles. The proposed cycle representation is a natural way to encode a strongly connected digraph, where a node is reachable to every other node by traversing a sequence of edges. The cycle representation also allows us to establish a contraction digraph model, where a path visiting a series of nodes is modelled as one edge. If the topological complexity of the candidate GN is moderate, its contraction model includes a limited number of nodes and we can drastically save the computational effort in evaluating the feasibility of the GN. The proposed GA works with special operators, which are well suited for the cycle representation. We empirically compared the proposed GA to the conventional ones for...
various problem instances. The numerical results exhibit the condition of the GNDP, where the merits of the proposed GA are fully utilized.

We organize this paper as follows. In Section II, we describe PRT system and formulate the GNDP. In Section III, we define the cycle representation and address the advantages of the contraction model. In Section IV, we propose a GA working with the cycle representation to solve the GNDP. In Section V, we empirically compare the proposed GA to the conventional GAs to show when the proposed GA outperforms the conventional ones. In Section VI, we draw the conclusions of this paper.

II. PERSONAL RAPID TRANSIT

A. System Description

Two main components of PRT are GN and vehicles. The GN consists of stations where passengers get in or off the vehicles and guideway links that provide one-way traffics between two stations. We can place two opposite directed guideway links between a station pair but not necessarily. A station has two parallel lines, one of which is a station siding and the other is a through line.

When a passenger party get in an empty vehicle, their destination encoded in a boarding pass is detected by the vehicle. Due to the multi-hop structure of the GN, the vehicle departing from the origin station may pass one or more intermediate stations to reach its destination. Meanwhile, the vehicle makes no stop at the intermediate stations by passing through the lines.

A vehicle runs with a constant speed except around the station siding and guideway merging point. The minimum headway between two subsequent vehicles are predetermined and the number of vehicles that can pass through a point on the guideway link is limited. This acts as a traffic capacity of the guideway link.

B. Problem Definition

The GN can be modelled as a subgraph of a complete digraph $G_C = (N,E)$, where $N$ is the set of nodes representing $|N|$ stations and $E = \{(i,j) \mid i,j \in N\}$ is the set of directed links representing all the guideway links between two stations in $N$. Given $G_C$, the objective of the GNDP is to find the shortest-length GN topology that satisfies the connectivity and traffic constraints. To have a formal description of the problem, let $d_{i,j}$ be the length of the guideway link between two nodes $i,j \in N$, $T = (t_{i,j}) \in \Re_{+}^{N \times N}$ be the O-D matrix representing the worst-case inter-station traffics, $t_{\text{CAP}}$ be the traffic capacity of the guideway link, and $X = (x_{i,j}) \in \{0,1\}^{N \times N}$ be the binary node adjacency matrix representing the presence of edges in a candidate GN. Edge $(i,j)$ exists in the GN if and only if $x_{i,j}$ is one.

Now, the GNDP can be formulated as follows:

**Given:** $N$, $\{d_{i,j}\}_{i,j \in N}$, $T$, and $t_{\text{CAP}}$

**Over:** $X$

**Minimize:** $c(X) := \sum_{i,j \in N} d_{i,j}x_{i,j}$

**Subject to:** connectivity and traffic constraints

In a practical situation, it is difficult to determine $d_{i,j}$ since the actual guideway link topology hinges on the curvature constraints for safety and existing obstacles such as buildings and hills. Moreover, a GN may have non-station nodes. Instead of placing two guideway links from station A to B and A to C, we can place a non-station node D within the triangle region formed by three vertices A, B, and C so that the traffics from A could follow the guideway link from A to D and then split into two links toward B and C. This will save the total guideway link length by sacrificing trip time.

To simplify the problem, we assume that there exist no non-station nodes and $d_{i,j}$ is the Euclidean distance between stations $i$ and $j$. We also assume that the incoming and outgoing traffics of a station are the same (a station has no vehicle buffer). The vehicle traffic follows the shortest-length route from the origin station to the destination since it is optimal in trip time and energy consumption.

C. Constraints

Two constraints, connectivity and traffic capacity, are involved in the GNDP problem. Firstly, the GN must be strongly connected to provide the accessibility between every station pair. The connectivity of a GN can be evaluated while we seek for the shortest-length routes over the GN. If an all-pair shortest-path algorithm such as Floyd-Warshall algorithm fails to find a route from station $i$ to $j$ over a GN, $j$ is disconnected from $i$ and the GN is infeasible.

Given the shortest-length routes between every station pair, we can also determine the worst-case traffic through the most congested point of the GN. This can be done by aggregating all the worst-case traffics supposed to pass through guideway links and picking up the largest value among them. Suppose that the shortest-length routes over $X$ are represented by route vectors $r_{i,j}(X) = (r_{i,j,k}(X)) \in \{0,1\}^n$ for $i,j \in N$ and $i \neq j$, where $i$ and $j$ represent the indices of origin and destination stations, respectively, and $k$ is the index of an intermediate station. If the shortest-distance route from station $i$ to $j$ passes station $k$, then $r_{i,j,k}(X)$ is one; otherwise, zero.

Let $\tau_{k}(X)$ be the worst-case traffic estimate at the exit point of station $k$ in $X$. We have

$$\tau_{k}(X) = \sum_{i \in N} t_{k,i} + \sum_{i,j \in N, i \neq j} r_{i,j,k}(X)t_{i,j}.$$ 

The first term on the righthand side represents the worst-case traffic demand departing from the siding of station $k$ while the second term represents the worst-case traffic estimate passing through station $k$. Since every guideway link traffic in $X$ is deviated from $\tau_{k}(X)$ for $k \in N$, the worst-case traffic estimate through the most congested point will be $\max_{k \in N} \tau_{k}(X)$. This value must be no greater than the link traffic capacity $t_{\text{CAP}}$, which is determined by minimum headway requirement.
III. CYCLE REPRESENTATION

A. Definition

To represent a subgraph $G$ of $G_C$ that connects every node in $N$, we use the cycle representation defined as follows:

$$G = \{\psi_1, \ldots, \psi_L\},$$

where $\psi_l$ for $l \in \{1, \ldots, L\}$ represents a simple cycle that sequentially visits $n_l$ nodes in $N$. A cycle $\psi_l$ is represented by a sequence of $n_l$ node indices ($n_l \geq 2$):

$$\psi_l = (\pi_l(1), \ldots, \pi_l(n_l)).$$

The $\psi_l$ defined as (2) is composed of $n_l$ edges, $(\pi_l(1), \pi_l(2)), (\pi_l(2), \pi_l(3)), \ldots, (\pi_l(n_l), \pi_l(1))$. Note that a cycle is simple and thus has no repeated node on it. A cycle with repeated nodes can be represented by two or more simple cycles. By taking the union of all the edges in $\psi_1, \ldots, \psi_L$, a candidate digraph $G$ is built. Two or more different cycles may contain the same edge, but they yield one edge in $G$.

For example, consider three cycles over seven nodes depicted on the left-hand side of Fig. 1. The three cycles are represented by

$$\psi_1 = (1, 2, 3),$$
$$\psi_2 = (1, 3, 4, 7),$$
$$\psi_3 = (5, 6, 7)$$

and generate 10 edges: $(1, 2), (2, 3), (3, 1), (1, 3), (3, 4), (4, 7), (7, 1), (5, 6), (6, 7)$, and $(7, 5)$. These edges form a seven-node digraph depicted on the right-hand side of Fig. 1.

Since a cycle is the minimum unit to guarantee the connectivity of a group of nodes, aggregating one or more cycles is a natural way to represent a connected digraph. The completeness of the cycle representation can be easily proved:

Lemma 1: The cycle representation defined in (1) can represent every strongly connected digraph with a finite node set.

Proof: Let $i$ be a node in a connected digraph. Since the digraph has a finite node set $N$, $i$ has indegree and outdegree no greater than $|N| - 1$. Let $j$ be a direct successor node of $i$ through the edge $(i, j)$. Due to the connectivity of the digraph, $j$ has a certain path to $i$ and the edge $(i, j)$ can be covered by the cycle obtained by concatenating the path and $(i, j)$. Other outgoing and incoming edges of $i$ can be covered by finite cycles in the same way and repeating this procedure to other nodes in $N$ proves that a set of finite cycles covers all the edges in the digraph.

Another advantage of the cycle representation is its high solution locality. This enables us to design efficient initialization and variation operators for the GA. Moreover, it is straightforward to generate the contraction model from the candidate GN represented by cycles as described in the following subsection.

B. Contraction Model

The time complexity of the feasibility evaluation is $O(|N|^2)$, which is consumed for all-pair shortest-path search. If hundreds or more stations are given, the feasibility evaluation of a candidate GN acts as the main computational bottleneck of the GA. Using the contraction model, however, we can drastically reduce the number of nodes involved in the all-pair shortest-path search if the given network has moderate topological complexity.

We check the feasibility of a given GN $G$ by conducting the following four steps:

1) Check if the cycles composing $G$ visit all the nodes in $N$. The worst-case time complexity of this step is $O(|N|L)$.
2) Evaluate the connectivity of $G$ by checking if there exist no isolated cycle groups. The worst-case complexity of this step is $O(|N|L)$ or $O(L^3)$.
3) Establish the contraction model $G_J$ of $G$. The time complexity for this step is $O(|N|L)$.
4) Evaluate the traffic feasibility of $G$ using $G_J$ as addressed in Section III-C. The worst-case complexity for this step is either $O(|N|^2)$ or $O(|N_J|^2)$.

If $G$ violates the condition described in Step 1 or 2, $G$ is disconnected and further feasibility check is meaningless. With the contraction model, the overall time complexity of the feasibility check is reduced from $O(|N|^3)$ to $O(|N|L)$, $O(L^3)$, $O(|N|^2)$, or $O(|N_J|^3)$.

The contraction model $G_J$ of $G$ is established as described below. We create the node adjacency matrix of $G$ and count the indegree and outdegree of every node. If a certain node has indegree or outdegree of two or more, it is entitled a junction node and copied to $N_J$. Identification of $N_J$ can be done while conducting Step 2. If $|N_J| = 0$, we have one Hamiltonian cycle visiting all the nodes and two arbitrary nodes in $N$ are copied to $N_J$. If $|N_J| > 0$, we have multiple cycles and a cycle can be viewed as a series of paths connecting two junction nodes. A path starts from a tail junction node, traverses zero or more non- junction nodes, and reaches a head junction node. The path length is defined as the length sum of all the guideway links and sidings in the
Fig. 2. Junction-node graph obtained from the three cycles in Fig. 1 (Double circles: junction nodes; arrows: paths; circles: non-junction nodes on paths).

Fig. 3. Pseudo code of the traffic computation algorithm based on a junction-node graph model.

C. Traffic Estimate

Fig. 3 outlines the pseudo code of the traffic estimate algorithm based on $G_J$, which consists of four procedures. Given $G_J$ and $T$, the first procedure $\text{Aggregate}_\text{traffic}$ generates inter-junction node traffics in the form of $T_J \in \mathfrak{N}[N_J \times [N_J]]$. Given a non-junction node $i$, let $i_H$, $i_T \in N_J$ be its head and tail junction nodes. If $i \in N_J$, $i_H$ and $i_T$ become $i$ itself. At the initial stage of $\text{Aggregate}_\text{traffic}$, $T_J$ is initialized as a zero matrix. For each ordered pair of nodes $i, j \in N_J$, we add $t_{i,j}$ to an element of $T_J$ corresponding to the ordered pair of $i_H$ and $i_T$ since the traffic from $i$ to $j$ passes through $i_H$ as the first junction node on its route and $j_T$ as the last. In Fig. 2, for example, the traffic from node 2 to 5 will pass 3 as the first junction node and 7 as the last. So, $t_{2,5}$ will be added to the element of $T_J$ corresponding to the ordered node pair $(3, 7)$. The time complexity of $\text{Aggregate}_\text{traffic}$ is $O(|N_J|^2)$.

The second procedure $\text{Generate}_\text{route}$ generates the all-pair shortest-length routes for $N_J$ with a time complexity of $O(|N_J|^3)$. Based on $T_J$ and the route information, the third procedure $\text{Compute}_\text{junction}_\text{node}_\text{traffic}$ calculates the traffic at every junction node as the sum of the traffic departing from it and the traffic passing through it. The last procedure $\text{Compensate}_\text{traffic}$ subtracts duplicate traffics from the junction node traffics. The duplication occurs from the traffic arriving at the junction node and the traffic within a path. Since the incoming and outgoing traffics of a node are the same and we only count the outgoing traffic, we must compensate the traffic arriving at the junction node. The traffics within a path does not affect the junction node traffic and must be subtracted. For example, the traffic from node 5 to 6 in Fig. 2 must be subtracted during $\text{Compensate}_\text{traffic}$.

IV. GENETIC ALGORITHM

To tackle the GNDP, we develop a steady-state GA with the cycle representation. The pseudo code of the GA is outlined in Fig. 4, where $g$ denotes the generation index. $P_g$ is the population at generation $g$. $P_{1}$ and $P_{2}$ are two parents selected to breed the child $I_C$, and $g_f$ is a termination generation. Each individual in $P_g$ represents a cycle set that generates a candidate digraph as defined in (1). According to the steady-state selection scheme, only one child is created and competes with the individuals in the current population to survive to the next generation. The detail procedures of the GA shall be described in the following subsections.

A. Initialization

The initialization procedure (Line 2 in Fig. 4) generates $|P|$ random individuals composing $P_0$. While generating an individual, we iteratively create a new cycle and add it to the individual until all the nodes in $N$ are visited by at least one cycle. When creating a new cycle $\psi$, we first generate a uniform random integer $u \in \{2, \ldots, |N|\}$, which will be the number of nodes $\psi$ visit. We randomly pick one node that is not visited by any cycle yet and add it to an empty node set $\eta$. Then, we randomly choose $u - 1$ nodes in $N$ and add them to $\eta$ so that there exists no duplicated nodes in $\eta$.

To determine the sequence of nodes $\psi$ visits, we apply the nearest neighbor algorithm whose time complexity is $O(|\eta|^2)$. In a reasonable computation time, the nearest neighbor algorithm generates a cycle whose length is comparable to the shortest cycle visiting all the nodes in $\eta$. We need not generate the shortest cycle over $\eta$ since the union of shortest cycles does not guarantee the optimality of the GNDP due to the traffic constraint. To secure a more chance to generate various cycles over the same $\eta$, we randomly choose the starting node of the nearest neighbor algorithm.
ψ adds a new cycle is the case, the repair procedure (Line 4 in Fig. 4) repeatedly more cycles in an individual of the longest edge in Fig. 5. Crossover operation over two parents, represented by cycles.

B. Repair Algorithm

Even when all the nodes in N are visited by one or more cycles in an individual I, two nodes in N may be disconnected due to the existence of isolated cycles. If this is the case, the repair procedure (Line 4 in Fig. 4) repeatedly adds a new cycle ψ to I until I becomes connected. When creating ψ, we first choose the size of ψ as a uniform random integer in \( \{2, \ldots, |N|\} \). We then pick a node in a isolated cycle and add it to ψ. The other nodes of ψ are randomly chosen from N avoiding duplication and the nearest neighbor algorithm is applied to ψ. We check the connectivity of I based on the cycle adjacency graph of I.

C. Evaluation

During the evaluation procedure of I (Line 5 in Fig. 4), we evaluate both the cost \( c(I) \) and maximum node traffic \( \tau_{\text{MAX}}(I) \). We compute \( c(I) \) by summing the lengths of all the edges in I. To handle the traffic constraint, we compute \( \tau_{\text{MAX}}(I) \) as described in Section III-C. The objective function \( f(I) \) is given as the sum of \( c(I) \) and the penalty term regarding the traffic constraint:

\[
f(I) = \begin{cases} 
  c(I) & \text{if } \tau_{\text{MAX}}(I) \leq t_{\text{CAP}}, \\
  c(I) + \frac{K - \tau_{\text{MAX}}(I) - t_{\text{CAP}}}{t_{\text{CAP}}} & \text{otherwise,}
\end{cases}
\]

where \( K > 0 \) is a penalty constant and \( d_{\text{max}} \) is the length of the longest edge in \( G_C \).

D. Parent Selection

To choose two parent individuals \( I_{P1} \) and \( I_{P2} \) from \( P_g \), we conduct tournament selection with a tournament size \( k_p \) (Line 8 in Fig. 4). That is, we randomly choose \( k_p \) individuals from \( P_g \) and select the individual with the least \( f \) values as \( I_{P1} \). The same procedure is used to select \( I_{P2} \).

E. Crossover

The child individual \( I_C \) is created from the crossover operation over \( I_{P1} \) and \( I_{P2} \) (Line 9 in Fig. 4). Several crossover operators have been reported to handle permutation-based representation. Such operators include, for example, order crossover [17] and partially mapped crossover [18]. Although these algorithms have been successfully applied to resolve a TSP, they are not well suited for the cycle representation, where multiple cycles are involved to represent one individual.

In the presented GA, the crossover operation is conducted with a probability \( p_C \in [0, 1] \) and the parent \( I_{P1} \) is simply copied to \( I_C \) with a probability of \( 1 - p_C \). The crossover operation is done by considering a cycle as the minimum unit for information swapping. Each cycle in \( I_{P1} \) and \( I_{P2} \) is copied to \( I_C \) with a probability 0.5. Therefore, \( I_C \) created from the crossover operation may contain no cycles or all the cycles in \( I_{P1} \) and \( I_{P2} \) in the extreme cases. Fig. 5 illustrates how the crossover works. A disconnected child generated from the crossover operation will be repaired while undergoing mutation operation.

F. Mutation

Contrary to the standard mutation operation for the edge representation, the cycle representation enables us to develop various mutation operators maintaining the connectivity of the child. The proposed mutation operation (Line 10 in Fig. 4) comprises two phases. In the first phase, we perform cycle-level mutation operations such as cycle-merging or cycle-splitting. In the second phase, we mutate each cycle with cycle-resizing or sequence-changing operations.

Given \( I_C \), the four subfunctions of the mutation operation are applied in turn. The cycle-merging function is executed with a probability \( p_M \in [0, 1] \). It randomly selects two cycles in \( I_C \) and takes the union of the nodes visited by the two cycles. By applying the nearest neighborhood algorithm to the union, we generate a new cycle, which replaces the two cycles. Fig. 6 illustrates how the cycle-merging function works.

The cycle-splitting function is activated with a probability \( p_S \in [0, 1] \). It randomly selects a cycle \( \psi \) in \( I_C \) and builds a node set \( \eta \) with all the nodes \( \psi \) visits. We then generate a uniform random integer \( u \in [1, \ldots, |\eta| - 2] \) and remove \( u \) nodes in \( \eta \) randomly. By applying the nearest neighbor to \( \eta \), we create a new cycle and add it to \( I_C \). Since \( \psi \) still remains in \( I_C \), one or more edges in the new cycle work as shortcuts of \( \psi \). Fig. 7 depicts the cycle-splitting operation.

The cycle-resizing function is applied to each cycle \( \psi \) in \( I_C \) with a probability \( p_R \in [0, 1] \). Once applied, the cycle-resizing function either adds a new node to \( \psi \) or removes a node from \( \psi \) with the same probability 0.5. If \( \psi \) visits 2 (or \(|N|\)) nodes, this function does not remove (or add) a node.

---

Fig. 5. Crossover operation over two parents, \( I_{P1} \) and \( I_{P2} \) represented by cycles.

Fig. 6. Cycle-merging function.
The nearest neighbor algorithm is then applied to the resized cycles. Lastly, the sequence-changing algorithm is applied to each cycle with a probability \( p_0 \in [0,1] \). The sequence-changing function is a 2-Opt algorithm [19]: It randomly chooses two edges in a cycle and deletes them to generate two separate paths. The two paths are reconnected in the other possible way by reversing the node sequence of one path. The sequence-changing function is essential since it is the only mechanism to escape from a suboptimal solution generated by the nearest neighbor algorithm. Fig. 8 illustrates how the sequence-changing function works.

**G. New Generation**

After repairing and evaluating \( I_C \) (Line 11 and 12 in Fig. 4), we form \( P_{g+1} \) by replacing the worst individual \( I_{\text{wor}} \) in \( P_g \) with \( I_C \) if \( f(I_C) < f(I_{\text{wor}}) \) (Line 13 in Fig. 4). The evolutionary process repeats until \( g \) reaches the termination generation \( g_T \).

**V. NUMERICAL RESULTS**

To verify the efficiency of the cycle representation, numerical experiments were performed. We compared the solution quality and computation time of the proposed GA to those of two conventional GAs with the edge representation. Problem instances with a realistic GN size were generated for the numerical experiments. While generating a GNDP instance, we chose the node position randomly and uniformly in a unit square. The edge length \( d_{i,j} \) was given as the Euclidean distance between node \( i \) and \( j \). The inter-node traffic \( t_{i,j} \) was initialized as a random number in \([0,1]\). We then updated \( t_{i,j} \) using northwest corner method [20] to compensate the differences in incoming and outgoing traffics of nodes.

**B. Problem Instances**

We used four GNDP instances listed in Table I for the numerical experiments. While generating a GNDP instance, we chose the node position randomly and uniformly in a unit square. The edge length \( d_{i,j} \) was given as the Euclidean distance between node \( i \) and \( j \). The inter-node traffic \( t_{i,j} \) was initialized as a random number in \([0,1]\). We then updated \( t_{i,j} \) using northwest corner method [20] to compensate the differences in incoming and outgoing traffics of nodes.

**C. Comparison on Solution Quality**

We first conducted 20 independent runs of the three GAs over GNDP20 with \( t_{\text{CAP}} = 100 \). Table II lists the statistics of elitists’ \( f \) values obtained at \( g = g_T \) for the three GAs. It can be seen that the GAC consistently yielded better solution quality than the GAE1 and GAE2 did. Fig 9 depicts the improvement histories of elitists’ \( f \) values over \( g \) for the best trials of the three GAs. At the initial stage of the evolution, the GAE1 and GAE2 worked with expensive GN candidates. The relationship between \( p_1 \) and the associated solution quality was negligible as shown in Table III. On the other hand, the GAC mostly generated connected and cheap solutions.

**TABLE I**

<table>
<thead>
<tr>
<th>Name</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GNDP20</td>
<td>20</td>
<td>380</td>
</tr>
<tr>
<td>GNDP30</td>
<td>30</td>
<td>850</td>
</tr>
<tr>
<td>GNDP40</td>
<td>40</td>
<td>1560</td>
</tr>
<tr>
<td>GNDP50</td>
<td>50</td>
<td>2450</td>
</tr>
</tbody>
</table>

**Fig. 7. Cycle-splitting function.**

**Fig. 8. Sequence-changing function.**
TABLE II
AVERAGE ELITES’ J VALUES AT g = gT OF THE THREE GAS EXECUTED OVER GNDP20 WITH tCAP = 100.

<table>
<thead>
<tr>
<th></th>
<th>GAC Mean</th>
<th>GAE1 Mean</th>
<th>GAE2 Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Std. dev</td>
<td>Std. dev</td>
<td>Std. dev</td>
</tr>
<tr>
<td></td>
<td>0.18</td>
<td>0.47</td>
<td>0.28</td>
</tr>
</tbody>
</table>

Fig. 9. Improvement history of elites’ j values over g for the best trials of the three GAs (solid: GAC; dotted: GAE1; dashed: GAE2).

GN candidates during the initialization procedure and found a good solution even at the early stage of the evolution.

D. Comparison on Computation Time

To compare the computational loads of the three GAs, we recorded the elapse times of the 60 GA runs executed in Section V-C. Table IV shows the statistics of the elapse times at g = gT. The high-fitness GN candidates had no or a few junction nodes and we observed considerable difference in computation time between GAC and others.

To examine the relationship between the number of junction nodes in solution candidates and the associated performance of the GAC, we repeated the experiment after reducing tCAP to 75. As the node traffic capacity decreases, we need more edges and junction nodes in the GN to meet the traffic constraint. Table V lists the solution qualities and computation times of the three GAs executed over GNDP20 with tCAP = 75. The solution quality of the GAC was still better than those of GAE1 and GAE2, but the computation time of the GAC became comparable to that of GAE1. The number of junction nodes in the best solution reached around a half |N|, the solution quality of the GAC was better than the others. However, the computation times of the GAE1 for GNDP40 and GNDP50 became shorter than those of the GAC.

VI. Conclusions

In this paper, we proposed a GA with the cycle representation to address the GNDP. The cycle representation is a natural way to represent a GN composed of multiple circulators and enables us to establish the contraction model, where its feasibility can be checked with less computational effort. To solve the solution candidates represented by cycle sets, new initialization, repair, crossover, and mutation operators were proposed. We empirically compared the solution quality and computation time of the proposed GA to the conventional edge represented GAs. The results suggest that the proposed GA outperforms the conventional GAs if the optimal solution has reasonable topological complexity.

We could improve the proposed GA by considering more sophisticated cycle initialization and tuning mechanisms. In this study, we used the nearest neighbor algorithm to initialize a cycle and the sequence-changing mutation function to tune it. For better optimization of each cycle, we may take advantage of a local search operator [21], [22]. For example, we can apply the sequence-changing function to a cycle repeatedly and accept the change if it gives better solution quality. Another existing technique for cycle generation and tuning is an ant algorithm [23]. The original ant algorithm was developed to solve a TSP, where every node in N should be visited once. If we allow an ant to revisit nodes, the ant may generate multiple simple cycles representing a candidate digraph. Sending multiple ants is an alternative way to generate multiple cycles. To implement this hybridization, we should develop effective pheromone update strategy and ant movement algorithm well suited for the GNDP.

For GNDP30, GNDP40, and GNDP50, we tested the three GAs with different tCAP values and obtained the results listed in Table VII and VIII. As expected, the GAC consistently outperformed the GAE1 and GAE2 if tCAP was sufficiently large. Even when tCAP decreased and the number of junction nodes in the best solution reached around a half |N|, the solution quality of the GAC was better than the others. However, the computation times of the GAE1 for GNDP40 and GNDP50 became shorter than those of the GAC.
TABLE VII
AVERAGE ELITES’ F VALUES AT g = gT OF THE THREE GAS EXECUTED OVER GNDP30, GNDP40, AND GNDP50 (THE NUMBER OF JUNCTION NODES WAS COUNTED FOR THE BEST SOLUTION OBTAINED FROM THE GAC RUNS).

<table>
<thead>
<tr>
<th>Problem instance</th>
<th>GAC</th>
<th>GAE1</th>
<th>GAE2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std. dev.</td>
<td>Mean</td>
</tr>
<tr>
<td>GNDP30</td>
<td>150</td>
<td>6.45</td>
<td>0.39</td>
</tr>
<tr>
<td>GNDP40</td>
<td>200</td>
<td>5.77</td>
<td>0.33</td>
</tr>
<tr>
<td>GNDP50</td>
<td>100</td>
<td>10.47</td>
<td>0.52</td>
</tr>
</tbody>
</table>

TABLE V
AVERAGE ELITES’ F VALUES AT g = gT AND CPU SECONDS OF THE THREE GAS EXECUTED OVER GNDP20 WITH tCAP = 75.

<table>
<thead>
<tr>
<th>Problem instance</th>
<th>GAE</th>
<th>GAE1</th>
<th>GAE2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std. dev.</td>
<td>Mean</td>
</tr>
<tr>
<td>Average f value</td>
<td>5.65</td>
<td>0.22</td>
<td>6.45</td>
</tr>
<tr>
<td>CPU seconds</td>
<td>411</td>
<td>36</td>
<td>433</td>
</tr>
</tbody>
</table>

TABLE VI
AVERAGE ELITES’ F VALUES AT g = gT AND CPU SECONDS OF THE THREE GAS EXECUTED OVER GNDP20 WITH tCAP = 50.

<table>
<thead>
<tr>
<th>Problem instance</th>
<th>GAE</th>
<th>GAE1</th>
<th>GAE2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std. dev.</td>
<td>Mean</td>
</tr>
<tr>
<td>Average f value</td>
<td>9.58</td>
<td>0.45</td>
<td>9.22</td>
</tr>
<tr>
<td>CPU seconds</td>
<td>804</td>
<td>65</td>
<td>441</td>
</tr>
</tbody>
</table>

REFERENCES


