Algorithmica
https://doi.org/10.1007/s00453-018-0491-9

Modular Circulation and Applications to Traffic Management

Philip Dasler[1] · **David M. Mount**[1]

Received: 13 September 2017 / Accepted: 27 July 2018

Abstract
We introduce a variant of the well-known minimum-cost circulation problem in directed networks, where vertex demand values are taken from the integers modulo λ, for some integer $\lambda \geq 2$. More formally, given a directed network $G = (V, E)$, each of whose edges is associated with a weight and each of whose vertices is associated with a demand taken over the integers modulo λ, the objective is to compute a flow of minimum weight that satisfies all the vertex demands modulo λ. This problem is motivated by a problem of computing a periodic schedule for traffic lights in an urban transportation network that minimizes the total delay time of vehicles. We show that this modular circulation problem is solvable in polynomial time when $\lambda = 2$ and that the problem is NP-hard when $\lambda = 3$. We also present a polynomial time algorithm that achieves a $4(\lambda - 1)$-approximation.

Keywords Network flows and circulations · Traffic management · Approximation algorithms · NP-hard problems

1 Introduction

Minimum (and maximum) cost network flows and the related concept of circulations are fundamental computational problems in discrete optimization. In this paper, we introduce a variant of the circulation problem, where vertex demand values are taken from the integers modulo λ, for some integer $\lambda \geq 2$. For example, if $\lambda = 10$ a vertex with demand 6 can be satisfied by any net incoming flow of 6, 16, 26 and so on or a net outgoing flow of 4, 14, 24, and so on. Our motivation in studying this problem

Research supported by NSF Grant CCF-1618866.

✉ Philip Dasler
daslerpc@cs.umd.edu

David M. Mount
mount@cs.umd.edu

[1] Department of Computer Science, University of Maryland, College Park, MD 20742, USA

Published online: 13 August 2018

stems from an application in synchronizing the traffic lights of an urban transportation system.

Throughout, let $G = (V, E)$ denote a directed graph, and let $\lambda \geq 2$ be an integer. Each edge $(u, v) \in E$ is associated with a nonnegative integer *weight*, $wt(u, v)$, and each vertex $u \in V$ is associated with a *demand*, $d(u)$, which is an integer drawn from $\mathbb{Z}_\lambda$, the integers modulo λ. Let f be an assignment of values from $\mathbb{Z}_\lambda$ to the edges of G. For each vertex $v \in V$, define

$$f_{\text{in}}(v) = \sum_{(u,v)\in E} f(u, v) \quad \text{and} \quad f_{\text{out}}(v) = \sum_{(v,w)\in E} f(v, w),$$

and define the *net flow* into a vertex v to be $f_{\text{in}}(v) - f_{\text{out}}(v)$. We say that f is a *circulation with λ-modular demands*, or *λ-CMD* for short, if it satisfies the *modular flow-balance constraints*, which state that for each $v \in V$,

$$f_{\text{in}}(v) - f_{\text{out}}(v) \equiv d(v) \pmod{\lambda}.$$

Observe that a demand of $d(v)$ is equivalent to the modular "supply" requirement that the net flow out of this vertex modulo λ is $\lambda - d(v)$.

Define the *cost* of a circulation f to be the weighted sum of the flow values on all the edges, that is,

$$\text{cost}(f) = \sum_{(u,v)\in E} wt(u, v) \cdot f(u, v).$$

Given a directed graph G and the vertex demands d, the *λ-CMD problem* is that of computing a λ-CMD of minimum cost. (Observe that there is no loss in generality in restricting the flow value on each edge to $\mathbb{Z}_\lambda$, since the cost could be reduced by subtracting λ from this value without affecting the flow's validity.)

The standard minimum-cost circulation problem (without the modular aspect) is well studied. We refer the reader to any of a number of standard sources on this topic, for example, [1,3,7,10,12,14]. In contrast, λ-CMD is complicated by the "wrap-around" effect due to the modular nature of the demand constraints. A vertex's demand of $d(u)$ units can be satisfied in the traditional manner by having a net incoming flow of $d(u)$, but it could also be met by generating a net outgoing flow of $\lambda - d(u)$ (not to mention all variants thereof that involve adding multiples of λ). Our main results are:

- 2-CMD can be solved exactly in polynomial time (see Sect. 4).
- λ-CMD is NP-hard, for $\lambda \geq 3$ (see Sect. 5).
- There is a polynomial time $4(\lambda - 1)$-approximation to λ-CMD (see Sect. 6).

In Sect. 2 we discuss the relevance of the λ-CMD problem to a traffic-management problem. In Sect. 3 we present some preliminary observations regarding this problem. In Sects. 4–6 we present each of our three main results.

2 Application to Traffic Management

Our motivation in studying the λ-CMD problem arises from an application in traffic management. In urban settings, intersections are the shared common resource between

vehicles traveling in different directions, and their control is essential to maximizing the utilization of a transportation network [15]. There are numerous approaches to modeling traffic flow and diverse computational approaches to solve and analyze the associated traffic management problems. Dresner and Stone [5] use a multi-agent systems approach, in which vehicles request a reservation from the intersection, which in turn allocates each vehicle a time slot. Vehicles must then alter their speed to cross the intersection during their reserved time. Yu and LaValle [17] draw a connection between multi-agent path planning on collision-free unit-distance graphs and network flows. Despite the popular interest in automated traffic systems, there has been relatively little work on this problem from the perspective of algorithm design.

In an earlier paper [4], we considered the problem of scheduling the movements of a collection of vehicles through a system of unregulated crossings. Our approach was based on the idealized assumption that the motion of individual vehicles in the system is controlled by a central server. A more practical approach is based on aggregating vehicles into groups, or *platoons*, and planning the motion of these groups [11,16].

We consider the problem in this aggregated form, but from a periodic perspective. Consider an urban transportation network consisting of a grid of horizontal and vertical roads as laid out on a map. Each pair of horizontal and vertical roads meets at a unique intersection controlled by a traffic light that alternates between horizontal and vertical traffic, such that the pattern repeats over a time interval λ. We assume throughout that λ has been discretized to a reasonably small integer value, say in terms of seconds or tens of seconds.

More formally, we say that a traffic-light schedule is λ-*periodic* if it repeats every λ time units. We consider a traffic management system of the foreseeable future where the traffic light schedule is transmitted to the vehicles, which in turn may adjust their speeds to avoid excessive waiting at intersections. While vehicles may turn at intersections, the schedule is designed to minimize the delay of straight-moving traffic.

To motivate the connection with modular circulations, consider a four-sided city block (see Fig. 1). Let a, b, c, and d denote the intersections, and let t_{ab}, t_{bc}, t_{cd}, t_{ad} denote the travel times between successive intersections along each road segment. (We ignore practical issues such as acceleration.) If the road segment is oriented counterclockwise around the block (as shown in our example), these travel times are positive, and otherwise they are negative. Suppose that the traffic-light schedule is

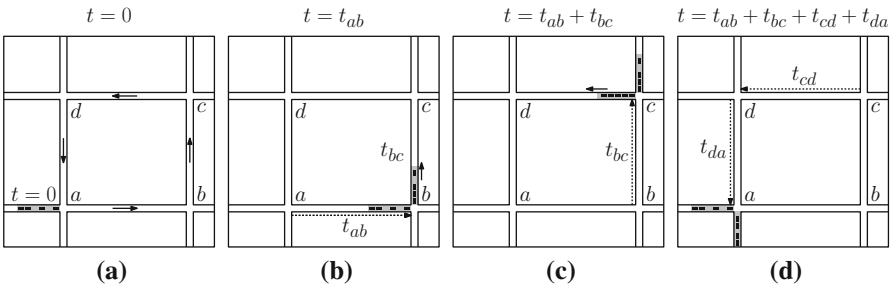

Fig. 1 Delay-free traffic-light schedule

λ-periodic, and that at time $t = t_a$ the light at intersection a transitions so that the eastbound traffic can move horizontally through the intersection (see Fig. 1a). In order for these vehicles to proceed without delay through intersection b, this light must transition from vertical to horizontal at time $t_b = t_a + t_{ab}$ (see Fig. 1b).

Reasoning analogously, for the other intersections, it follows that the vertical-to-horizontal transition times for intersections c and d are $t_c = t_b + t_{bc}$ and $t_d = t_c + t_{cd}$, respectively (see Fig. 1c). On returning to a (see Fig. 1d), we find that

$$t_a \equiv t_a + (t_{ab} + t_{bc} + t_{cd} + t_{da}) \pmod{\lambda}.$$

Thus, in order to achieve delay-free flow around the intersection in a λ-periodic context, we must satisfy the constraint

$$t_{ab} + t_{bc} + t_{cd} + t_{da} \equiv 0 \pmod{\lambda}.$$

While traffic-light scheduling is essentially cyclic in nature, we acknowledge that our model ignores many practical issues that arise in real traffic-management systems. Nonetheless, it is noteworthy that even distilled to its simplest elements, this problem is far from trivial. A natural question involves the choice of the integer parameter λ. This models the length of the time period shared by the synchronized traffic lights, but in what units is it expressed? Ideally, a traffic manager would like to have the greatest degree of flexibility in choosing λ, since delays must be rounded to integer multiples of λ. For example, if the traffic lights cycle every minute, then to achieve a resolution of seconds in the delay values, we would set $\lambda = 60$. Because our approximation bounds degrade as λ increases, it may be better to decrease the minimum time interval to, say, 15 seconds. Then we could set $\lambda = 60/15 = 4$, resulting in more accurate approximation bounds. The price paid is that delays would now be rounded up to multiples of 15-second intervals.

In an earlier paper we considered the special case where $\lambda = 2$ and opposite sides of the block have equal travel times [4]. We showed there that a simple alternating strategy is optimal. It is not reasonable, however, to assume that all the blocks in the city will satisfy this requirement. Furthermore, the assumption that $\lambda = 2$ does not admit a more nuanced timing of the lights. In order to handle the more general case, we introduce an (ideally small) nonnegative delay $\delta_{ij} \geq 0$ along each road segment ij. This yields the new constraint

$$(t_{ab} + \delta_{ab}) + (t_{bc} + \delta_{bc}) + (t_{cd} + \delta_{cd}) + (t_{da} + \delta_{da}) \equiv 0 \pmod{\lambda},$$

or equivalently, if we define $T = t_{ab} + t_{bc} + t_{cd} + t_{da}$ to be the sum of (signed) travel times of the road segments around this block, we have

$$\delta_{ab} + \delta_{bc} + \delta_{cd} + \delta_{da} \equiv -T \pmod{\lambda}. \tag{1}$$

The upshot is that if vehicles travel at a reduced speed so that the transit time along each of the road segments includes the associated delay, then the straight-line vehicular traffic along each road need never pause or wait at any traffic signal. The objective is

to minimize the sum of delay values over all the road segments in the network, which we refer to as the *total delay*.

More formally, the transportation network is modeled as a set of horizontal and vertical roads. This defines a directed grid graph whose vertices are the intersections, whose edges are the *road segments*, and whose (bounded) faces are the *blocks* of the city. For each pair of adjacent intersections i and j, let t_{ij} denote the delay-free travel time along this road segment. For each block u, define the *total signed travel time* about u to be the sum of the travel times for each of the road segments bounding u, where the travel time is counted positively if the segment is oriented counterclockwise about u and negatively otherwise. Let $T(u)$ denote this value modulo λ. A λ-*periodic traffic-light schedule* assigns a *delay* to each road segment so that for each block, these delays satisfy Eq. (1). The objective is to minimize the *total delay*, which is defined to be the sum of delays over all the segments in the network.

To express this in the form of an instance of λ-CMD, let $G = (V, E)$ denote the directed dual of the graph, by which we mean that the vertex set V consists of the city blocks, and there is a directed edge $(u, v) \in E$ if the two blocks are incident to a common road segment, and the direction of the road segment is counterclockwise about u (and hence, clockwise about v). The demand of each vertex u, denoted $d(u)$, is set to $T(u)$, and the weight of each edge is set to unity.

There remains one impediment to linking the λ-periodic traffic-light schedule and the λ-CMD problems. The issue is that the delay associated with any road segment (which may be as large as $\lambda - 1$) can be significantly larger than the time to traverse the road segment. If so, the capacity of the road segment to hold the vehicles that are waiting for the next signal may spill backwards and block the preceding intersection. In order to deal with this issue without complicating our model, we introduce the assumption that λ is smaller than the time to traverse any road segment. The link between the two problems is presented in the following lemma.

Lemma 2.1 *Given a transportation network and integer $\lambda \geq 2$, let G be the associated directed graph with vertex demands and edge weights as described above.*

(i) *If there exists a λ-periodic traffic-light schedule with total delay Δ, then there exists a λ-CMD for G of cost Δ.*
(ii) *If there exists a λ-CMD for G of cost Δ and for all road segments ij, $t_{ij} \geq \lambda$, then there exists a λ-periodic traffic-light schedule with total delay Δ.*

Proof To prove (i), consider any λ-periodic traffic-light schedule. Define a flow on G where the edge (u, v) carries flow equal to the delay on the associated road. The net flow into a vertex u of G is equal to the sum of delays for the road segments that are oriented clockwise about u and the sum of the negated delays for the road segments that are oriented counterclockwise. Given any block with (counterclockwise) intersections a, b, c, and d, the net flow into the associated node is $-(\delta_{ab} + \delta_{bc} + \delta_{cd} + \delta_{da})$. By the same reasoning behind Eq. (1), this sum is equivalent to $T(u)$ modulo λ. It follows directly that f is a λ-CMD. In both cases the cost is the sum of delays.

To prove (ii), let f denote any λ-CMD for the directed dual graph G. Fix any intersection and set its transition time to 0 modulo λ. The transition times for the remaining intersections can be set by a simple propagation process. In particular, if

the transition time of an intersection i is known, then the transition time of an adjacent intersection j is delayed relative to i (modulo λ) by δ_{ij}, which is the flow along the associated dual edge. By Eq. (1) it follows that whenever this propagation loops back to an intersection whose delay was set, the propagated transition time is equivalent (modulo λ) with the original transition time. By the assumption that $t_{ij} \geq \lambda$, the vehicles that entered this road segment during the last traffic-light cycle have sufficient space[1] that they can effectively "park" themselves within the road segment for δ_{ij} time units until moving through the next intersection. □

3 Preliminaries

In this section we present a few definitions and observations that will be used throughout the paper. Given an instance $G = (V, E)$ of the λ-CMD problem, consider any subset $V' \subseteq V$. Let $G' = (V', E')$ be the associated induced subgraph of G, and let $d(V')$ denote the sum of demands of all the nodes in V'. We refer to E' as the *internal edges* of this subgraph, and we refer to the edges of G that cross the cut $(V', V \setminus V')$ as the *interface*. Given such a subgraph and any flow f on G, define its *internal flow* to be only the flow on the internal edges, and define the *internal cost* to be the cost of the flow restricted to these edges. Define the *interface flow* and *interface cost* analogously for the interface edges. Define $f_{\text{in}}(V')$ to be the sum of flow values on the interface edges that are directed into V', and define $f_{\text{out}}(V')$ analogously for outward directed edges. The following lemma provides necessary and sufficient conditions for the existence of a λ-CMD.

Lemma 3.1 *Given an instance $G = (V, E)$ of the λ-CMD problem:*

(i) *For any induced subgraph $G' = (V', E')$ and any λ-CMD f, we have*

$$f_{\text{in}}(V') - f_{\text{out}}(V') \equiv d(V') \pmod{\lambda}.$$

(ii) *If G is weakly connected, then a λ-CMD exists for G if and only if $d(V) \equiv 0 \pmod{\lambda}$.*

Proof Assertion (i) follows by applying standard results on circulations to the modular context. The "only if" part of assertion (ii) is a special case of (i), where $G' = G$. To see the "if" part of assertion (ii), consider any path (ignoring the edge directions) between two vertices u and v of nonzero demand in G. For any x, $0 \leq x < \lambda$, we can push x units of flow from u to v by pushing x units of flow along each forward-directed edge of the path and $\lambda - x$ units of flow along each backward-directed edge. (Each intermediate vertex has a net incoming flow of either 0, λ or $-\lambda$, depending on the

[1] We are using the term "space" abstractly. To relate space and time, we would need to introduce a maximum speed limit, call it σ. Assuming vehicles move at their maximum speed, the total length of a vehicle platoon that can pass through any intersection in time λ is $\sigma\lambda$. To move these vehicles to the next intersection in time t_{ij} at full speed, the road segment between these intersections is of length σt_{ij}. Since $t_{ij} \geq \lambda$, we have $\sigma t_{ij} \geq \sigma\lambda$, and, irrespective of the choice of σ, it follows that there is sufficient space to park these vehicles, whatever their actual sizes may be.

orientation of the incident edges.) By repeating this, we can satisfy the demands of all the vertices in the graph. □

It follows from this lemma that the λ-CMD instance associated with any traffic-light scheduling problem has a solution. The reason is that each edge (u, v) contributes its travel time t_{uv} positively to $d(u)$ and negatively to $d(v)$, and therefore the sum of demands over all the vertices of the network is zero, irrespective of the travel times.

4 Polynomial Time Solution to 2-CMD

In this section we show that 2-CMD, which we also call *binary CMD*, can be solved in polynomial time by a reduction to minimum-cost matching in general graphs. Intuitively, the binary case is simpler because the edge directions are not significant. If a vertex is incident to an even number of flow-carrying edges (whether directed into or out of this vertex), then the net flow into this vertex modulo λ is zero, and otherwise it is one. Thus, solving the problem reduces to computing a minimum-cost set of paths that connect each pair of vertices of nonzero demand, which is essentially a minimum-cost perfect matching in a complete graph whose vertex set consists of the subset vertices of nonzero demand and whose edge weights are distances between vertices ignoring edge directions. The remainder of this section is devoted to providing a formal justification of this intuition.

Recall $G = (V, E)$ is a directed graph, and $d(v)$ denotes the demand of vertex v. Since $\lambda = 2$, for each $v \in V$, we have $d(v) \in \{0, 1\}$. Let $G' = (V, E')$ denote the graph on the same vertices as G but with directions removed from all the edges. We may assume that G' is connected, for otherwise it suffices to solve the problem independently on each connected component of G' and combine the results. We set the weight of each edge of G' to the weight of the corresponding edge of G. If there are two oppositely directed edges joining the same pair of vertices, the weight is set to the minimum of the two.

Let $U = U(G)$ denote the subset of vertices of V whose demand values are equal to 1. By Lemma 3.1(ii), we may assume that $d(V) \equiv 0 \pmod{\lambda}$. Therefore, $d(V)$ is even, which implies that $|U|$ is also even. For each $u, v \in U$, let $\pi(u, v)$ denote the shortest weight path between them in G', and let $wt(\pi(u, v))$ denote this weight. Define $\widehat{G} = (U, \widehat{E})$ to be a complete, undirected graph on the vertex set U, where for each $u, v \in U$, the weight of this edge is $wt(\pi(u, v))$. (This is well defined by our assumption that G' is connected.) Since $\widehat{G}$ is complete and has an even number of vertices, it has a perfect matching. The reduction of 2-CMD to the minimum-cost perfect matching problem is implied by the following lemma.

Lemma 4.1 *Given an instance $G = (V, E)$ of the 2-CMD problem, the minimum cost of any 2-CMD for G is equal to the minimum cost of a perfect matching in $\widehat{G}$.*

Proof We begin with the assertion that any 2-CMD f for G can be mapped to a collection of paths in the undirected graph G' corresponding to edges of f that carry nonzero flow, such that each $u \in U$ is incident to at least one such path. This is well known when dealing with traditional flows, but it does not generally hold for CMDs when $\lambda > 2$. We will show that it holds in the binary case.

To establish the assertion, let f denote a 2-CMD for G. Each edge of G carries either a flow of 0 or 1. Observe that the net flow into each vertex of U (respectively, $V \setminus U$) is odd (respectively, even). Consider the following process, which will output a collection of paths while gradually reducing f to an empty flow. Start with any vertex v that is incident to an odd number of flow-carrying edges. Compute any maximal path in G' by following edges that carry nonzero flow. (The edge directions are not important, because any incident edge that carries flow toggles the net flow's parity between even and odd.) Clearly, such a path must terminate at a different vertex u that is also incident to an odd number of flow-carrying edges, and hence is also in U. Output this path $\pi(u, v)$ from v to u and modify f by setting the flow values of the edges along this path to 0.

Observe that the resulting set Π of paths is incident to every vertex of U, and their total cost is equal to the cost of f. Also, the sum of edge weights along each path between vertices u and v is at most the cost of the shortest path between them, that is, $wt(\pi(u, v))$.

To establish the lemma, we apply the above path-reduction procedure. Each path between two vertices u and v corresponds to an edge of $\widehat{G}$. After reducing the flow values along the path between such a pair of vertices, these two vertices are now incident to an even number of edges carrying nonzero flow. Thus, each vertex of U will appear in exactly one of the output paths, implying that the final set of paths defines a perfect matching in $\widehat{G}$ (whose vertex set we recall is U). We have $\text{cost}(f) = \sum_{\pi(u,v)\in\Pi} wt(\pi(u, v))$, which is the cost of the perfect matching.

Conversely, given any perfect matching in $\widehat{G}$, we can generate a 2-CMD by pushing a single unit of flow along the edges of the shortest path between u and v, for each edge (u, v) of the matching. This is a valid 2-CMD, since each vertex of U is incident to an odd number of edges carrying one unit of flow, and each vertex of $V \setminus U$ is incident to an even number of edges carrying one unit of flow. Note that if paths share common edges, then the total flow value along this edge may exceed one, but if so it can be reduced to either 0 or 1. It follows that the cost of the 2-CMD is no larger than the cost of the matching. □

Our approach is similar to the classical solution to the Chinese-Postman Problem by Edmonds and Johnson [6]. Vertices of odd demand in our problem play the same role as vertices of odd degree in theirs. Since $\widehat{G}$ is dense, a minimum-cost perfect matching can be constructed in $O(|U|^3)$ time. The graph can be computed in $O(n^3)$ time, where $n = |V|$, by applying the Floyd–Warshall algorithm for computing shortest paths [3]. Thus, the overall running time is $O(n^3)$.

Theorem 4.1 *It is possible to solve the 2-CMD problem in $O(n^3)$ time on any instance $G = (V, E)$, where $n = |V|$.*

5 Hardness of λ-CMD

In this section, we present the following hardness result for λ-CMD.

Theorem 5.1 *For $\lambda \geq 3$, the λ-CMD problem is NP-hard.*

The reduction is from positive 1-in-3-SAT [9]. For the sake of brevity and simplicity, we show the proof for the case $\lambda = 3$ here, but the method easily generalizes (as described in "Appendix 1"). Let F denote a boolean formula in 3-CNF, where each literal is in positive form. Throughout, for $\alpha \in \{0, 1, \ldots, \lambda - 1\}$, we use the term α-vertex to denote a vertex whose demand is α. The reduction involves two principal components, a *variable gadget* which associates truth values with the variables of F and a *clause gadget* which enforces the condition that exactly one variable in each clause is assigned the value **True**.

5.1 Variable Gadget

Before discussing the general gadget, we describe a fundamental building block from which all variables will be constructed. The block consists of six vertices, three 1-vertices and three $(\lambda - 1)$-vertices (i.e., 2-vertices), connected together with edges as shown in Fig. 2a. Edges connecting 1-vertices have weight $wt(u, v) = 1.5$, while all other edges are of weight $wt(u, v) = 1$.

If a flow of 1 is sent from each 2-vertex to its connected 1-vertex, the 2-vertices overflow and all demands are satisfied with $cost(f) = 3$ (see Fig. 2b). This flow, in which there is no flow across the interface edges, represents a logical value of **False**.

If instead a flow of 1 is sent across the interface edges, then the demands of the 2-vertices are satisfied. A flow of 1 across each edge originating at the central 1-vertex will cause it to overflow and will satisfy each of the connected 1-vertices. This flow, in which each interface edge carries a flow of 1, represents a logical value of **True** and again has $cost(f) = 3$ (see Fig. 2c).

If every interface edge of a variable gadget carries the same flow and that flow is either 0 or 1, that variable is said to be *interface-consistent*. If all variables are consistent for a given flow, then that flow is said to be *variable-consistent*. Notice that both logical values above are realized via interface-consistent flows.

Lemma 5.1 *Given a fundamental block, a satisfying flow has cost* ≤ 3 *if and only if that flow is interface-consistent.*

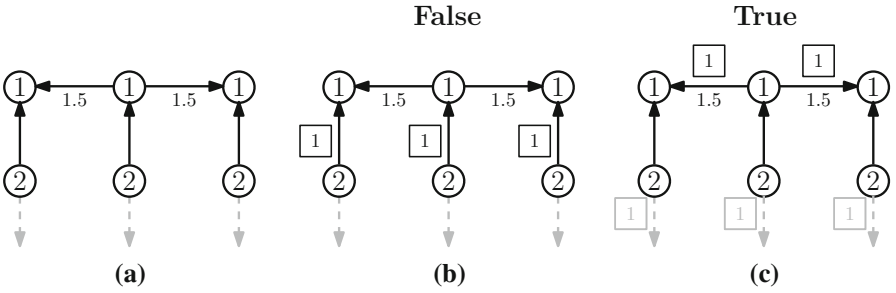

Fig. 2 **a** The fundamental building block used to build variable gadgets. Interface edges are dashed gray segments. **b**, **c** CMDs representing the assignment of **False** and **True** values, respectively, with the flow values in boxes

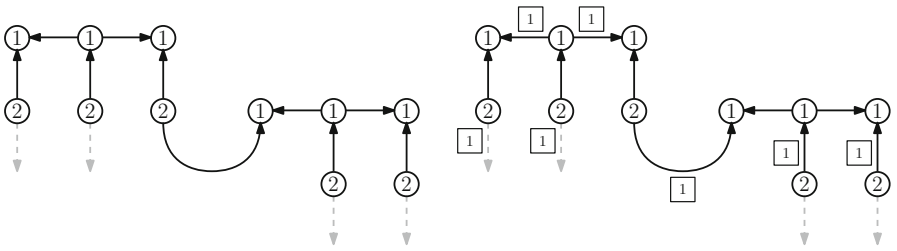

Fig. 3 Two fundamental blocks connected via a shared 2-vertex, with figure **b** showing a flow that is satisfying but not interface-consistent

Proof Each 2-vertex can only be satisfied by: (1) sending a flow of 1 across one of its edges or (2) sending a flow of 2 across both of its edges (in both cases the vertex's demand overflows).

In the second case, the 2-vertex sends a flow of 2 to its neighboring 1-vertex. As per Lemma 3.1(i), that vertex now requires a flow of 2 across its other edge in order to have its demand satisfied. Together these flows come at a cost of 5 (one of these edges has a weight of 1.5), therefor no 2-vertex may be satisfied by a flow greater than 1 without a cost greater than 3.

There exists a satisfying flow for a fundamental block if and only if the total flow across its interface edges is equivalent to $0 \pmod{\lambda}$ (see Lemma 3.1). Given this and the fact that no single interface flow may equal 2, the flows across the interface edges must either all be 0 or all be 1, i.e., the overall flow must be interface-consistent for it to be a satisfying flow. □

As variables may appear in multiple clauses, we need a mechanism by which existing variables can be expanded. For this, we create an expansion module, any number of which can be added to a variable so that there are three interface edges for each clause in which that variable appears.

To understand how this module functions, let us first look at the case when two fundamental blocks are connected together. This connection occurs through a shared 2-vertex, so that what was an interface edge for one block becomes the connection between a 2-vertex and 1-vertex in the other block (see Fig. 3). Recall that the value assignment of a variable is determined by the direction of flow from the 2-vertices, with flow along the interface representing **True** and internal flow (i.e., flow to the connected 1-vertices) representing **False**. Because the outgoing edges of the shared 2-vertex are simultaneously an interface edge of one block and an internal edge of the other, pushing a flow across either edge will assign opposing values to the blocks.

Knowing this, the module is constructed as a double-negative, ensuring that it is assigned the same value as the variable it extends. A fundamental block is used as a hub, and to this hub we attach two more fundamental blocks (see Fig. 4). When attached to a variable, this module creates four new interface edges and consumes one, thus extending the variable by three interface edges.

While the structure of the fundamental blocks is as described above, the weighting must be adjusted to maintain equal costs between the **True** and **False** states. Rather

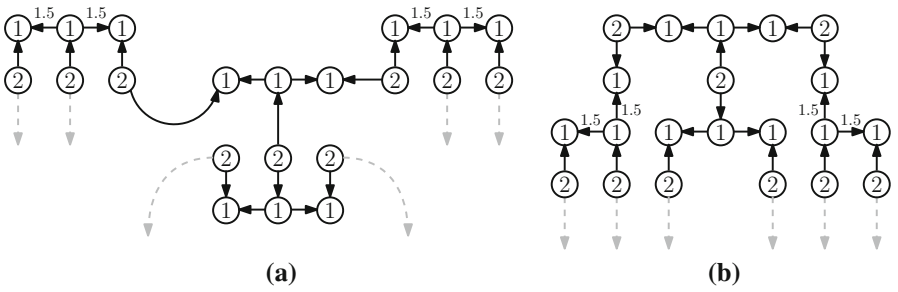

Fig. 4 **a** A fundamental block with a single expansion module attached. **b** The same structure rearranged to emphasize the two clause outputs, each with three interface edges

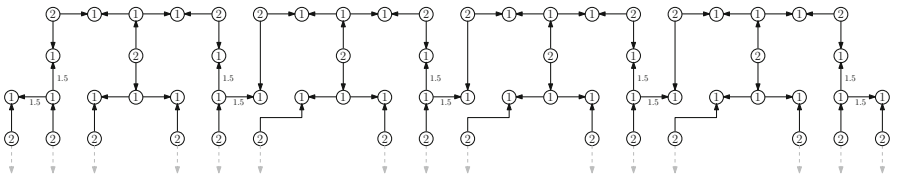

Fig. 5 An example of a large variable gadget with five clause outputs

than each fundamental block having two edges of weight 1.5, only the rightmost block in the expansion module has such edges; all others are of weight 1. In this way, the minimal cost of a consistent satisfying flow across the module is 8, regardless of the value assigned to the variable. This is easily verified by assigning a truth value to the gadget (fixing an interface-consistent flow on the interface edges of 0 or 1) and then traversing the structure, satisfying the demand in each vertex by assigning flow to its unused edge.

Given this, Lemma 5.1, and the fact that the expansion module is constructed from fundamental blocks, we have the following:

Lemma 5.2 *Given an expansion module, there exists an interface-consistent flow of internal cost* 8 *(in either the* **True** *or* **False** *cases), and any other satisfying flow has a strictly larger internal cost.*

To construct a gadget for a variable v_i, appearing in $c(v_i)$ clauses, begin with a fundamental block and connect $c(v_i) - 1$ expansion modules to it (see Fig. 5). Doing so provides $c(v_i)\lambda$ interface edges and yields the following result:

Lemma 5.3 *Given a variable gadget, there exists an interface-consistent flow of internal cost* $3 + 8[c(v_i) - 1]$ *(in either the* **True** *or* **False** *cases), and any other satisfying flow has a strictly larger internal cost.*

5.2 Clause Gadget

The basis for the clause gadget is a single 1-vertex with three incoming edges, one for each literal. These edges have a weight $wt(u, v) = \gamma$ and are connected to the

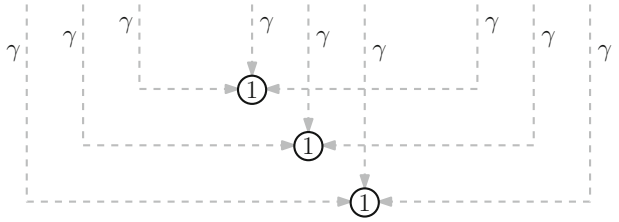

Fig. 6 A full clause gadget, with three inputs from each of three literals

appropriate variables as their outgoing interface edges. If a single literal is **True**, one of these edges will carry a flow of 1, satisfying the demand of the clause vertex. If more literals are **True**, the demand underflows and the vertex is left unsatisfied. It is possible to satisfy the vertex by creating flows on these edges greater than 1, but such flows can be made cost-prohibitive by setting the edge weights γ sufficiently high.

Recall that each variable gadget produces λ copies of its respective variable (three interface edges in this example) per clause in which it appears. Because of this, the clause gadget must also be created in triplicate. Every clause consists of three 1-vertices, each with an incoming edge from its three literals (see Fig. 6). Their weighting and behavior are as described above.

Since there are no internal edges in the clause gadgets, they do not contribute to the cost of the flow (but their interface edges will).

5.3 Final Construction

Each variable in F is represented by a fundamental block connected to $c(v_i) - 1$ expansion modules, creating $c(v_i)\lambda$ outputs. Thus, λ outputs are linked to each of the appropriate clause gadgets (see Fig. 7). The size of the variable gadget is a linear function of the number of clauses in which that variable appears and can thus be constructed in polynomial time.

If F is satisfiable, then a 3-CMD exists that is variable-consistent. In this case, each fundamental block incurs a cost of 3, and each expansion module incurs an additional cost of 8 for a flow representing a consistent truth value across its interface and the interfaces of the modules/fundamental block to which it is attached, as per Lemma 5.2.

For each clause, create λ 1-vertices, each connected to the clause's three literals by incoming edges. As there are no edges between these vertices, there is no flow possible within the clause gadget, resulting in an internal cost of 0. The size of the clause gadget is constant.

Finally, the flow on the edges between the variable gadgets and clause gadgets has yet to be counted as they are interface edges for both gadgets. Each clause gadget contains λ 1-vertices, with each receiving a flow of 1 across edges of weight γ. Thus, these add a cost of $3|C|\gamma$, where $|C|$ is the number of clauses in F.

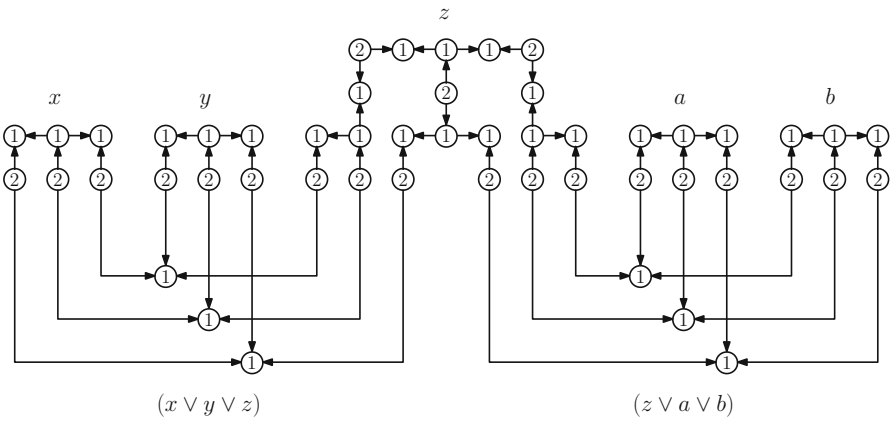

Fig. 7 An example of a reduction from the formula $F = (x \vee y \vee z) \wedge (z \vee a \vee b)$. Note that the weights have been removed for legibility

If F is not satisfiable, then some set of variables must have inconsistent outputs in order to create a valid CMD. As shown in Lemma 5.1, these inconsistencies will always lead to a strictly greater cost. Thus:

Lemma 5.4 *Given a positive boolean formula F in 3-CNF, in polynomial time it is possible to construct an instance of 3-CMD that has a satisfying flow with*

$$cost(f) \leq \sum_{v_i \in V} (3 + 8[c(v_i) - 1] + 3|C|\gamma)$$

if and only if F is 1-in-3 satisfiable.

6 Approximation Algorithm

In this section, we present a $4(\lambda - 1)$-factor approximation to the λ-CMD problem for $\lambda \geq 2$. Before presenting the algorithm, we introduce some terminology. Consider an instance $G = (V, E)$ of the λ-CMD problem, with vertex demands d. Let $G' = (V, E')$ be (as defined in Sect. 4) the undirected version of G. Let us assume that G' is connected. Let $U = U(G)$ denote the subset of vertices of V whose demand values are nonzero. Define $SMT(U)$ to be a *Steiner minimal tree* in G' whose terminal set is U (that is, a connected subgraph of G' of minimum weight that contains all the vertices of U).

As in Sect. 4, define $\widehat{G} = (U, \widehat{E})$ to be the complete, undirected graph over the vertex set U, where for each $u, v \in U$, the weight of this edge is the weight of a minimum weight path between u and v in G'. Given any $U \subseteq V$, let $MST(U)$ denote any minimum spanning tree on the subgraph of G' induced on U. From standard results on Steiner and minimum spanning trees we have the following.

Lemma 6.1 *For any $U \subseteq V$, $wt(MST(U)) \leq 2 \cdot wt(SMT(U))$.*

Define a *balanced partition* to be a partition $\{U_1, \ldots, U_k\}$ of U such that for $1 \leq i \leq k$, the total demand within U_i (that is, $d(U_i)$) is equivalent to zero modulo λ. By Lemma 3.1(ii), we may assume that $d(V) \equiv 0 \pmod{\lambda}$, and so there is always a trivial partition, namely $\{V\}$ itself. Define $\text{cost}(U_i)$ to be $\text{cost}(SMT(U_i))$, and define the *cost* of a balanced partition to be the sum of costs over its components. A *minimum balanced partition* for G is a balanced partition of minimum cost. The following lemma establishes the connection between balanced partitions and minimum modular circulations.

Lemma 6.2 *Consider an instance $G = (V, E)$ of λ-CMD. Let $\Psi = (U_1, \ldots, U_k)$ denote a minimum balanced partition of G, as defined above, and let f denote any minimum cost λ-CMD for G. Then* $\text{cost}(\Psi) \leq |f| \leq (\lambda - 1) \cdot \text{cost}(\Psi)$.

Proof Given any λ-CMD f for G, let $E'_f \subseteq E'$ denote the corresponding edges of the undirected graph G' that carry nonzero flow. This set of edges induces a set of connected components such that each vertex of U lies within one of these components. These components defines a partition of U. By Lemma 3.1(i), the sum of demands within each connected component is a multiple of λ, which implies that these connected components define a balanced partition for G, which we call Ψ_f. Since each edge that carries flow carries at least one unit of flow, it follows that $|f| \geq \text{cost}(\Psi_f)$. This establishes the lower bound on $|f|$.

In order to prove the upper bound, we will show how to map Ψ into a λ-CMD that satisfies the desired bound. Our construction builds a separate flow f_i for each subset U_i, and the final flow is simply the sum of all these flows. Let $T_i = SMT(U_i)$. It will simplify matters to first describe the construction in terms of the undirected graph G', and then modify it to apply to G.

Let us root T_i at any one of the vertices of U_i. The construction operates in a bottom-up manner by performing a post-order traversal of T_i. For each vertex u visited, let $f(u)$ denote the net flow into u from its children (or zero if u is a leaf). If $d(u) - f(u) < 0$) (this is a net-supply vertex) we direct $(d(u) - f(u)) \bmod \lambda$ units of flow from u to its parent, and if $d(u) - f(u) > 0$ (this is a net-demand vertex) we direct $(d(u) - f(u)) \bmod \lambda$ units of flow from its parent to u. Because the sum of demands within this component is a multiple of λ, the net flow into the root r must equal $d(r)$ modulo λ. It is easy to see that the net flow into every vertex of T_i is equivalent to zero modulo λ, and thus it is a valid λ-CMD. It follows directly that by applying the process to every component of Ψ we obtain a λ-CMD for G', which we denote by f'_Ψ. Because each edge carries a flow of at most $\lambda - 1$, we have $|f'_\Psi| \leq (\lambda - 1) \cdot \text{cost}(\Psi)$.

We can convert f'_Ψ to a λ-CMD for the directed graph G as follows. Consider any edge (u, v) of any subtree T_i where u is the child and v is the parent. If the direction of this edge in G matches the direction of the flow, then there is no change. Otherwise, we replace the flow value of x on this edge with the flow value of $\lambda - x$, essentially negating the value of the flow. It is easy to see that the resulting flow satisfies the modular balance constraints at each vertex, and therefore the result is a λ-CMD for G. By the same reasoning as above, $|f_\Psi| \leq (\lambda - 1) \cdot \text{cost}(\Psi)$. □

By the above lemma, it suffices to compute a balanced partition for G of low cost. We will present a simple approximation algorithm that outputs a balanced partition whose cost is within a factor of 4 of the optimum.

The construction begins with the metric closure $\widehat{G}$ defined above. In a manner similar to Kruskal's algorithm, we sort the edges of $\widehat{G}$ in increasing order, and start with each vertex of $\widehat{G}$ in a separate component. All these components are labeled as *active*. We process the edges one by one. Letting (u, v) denote the next edge being processed, if u and v are in distinct components, and both components are active, we merge these components into a single component. If the sum of the demands of the vertices within this component is equivalent to zero modulo λ, we label the resulting component as *finished*, and output its set of vertices. Because the total sum of demands of all the nodes is equivalent to zero modulo λ, it follows that every vertex is placed within a finished component, and therefore the algorithm produces a balanced partition of $\widehat{G}$ (and by extension, a balanced partition of G).

This algorithm has the same running time as Kruskal's algorithm. (Observe that we can associate each component with its sum of demands, thus enabling us to determine the sum of merged components in constant time.) The following lemma establishes the approximation factor for this construction.

Lemma 6.3 *Let Ψ' denote the balanced partition generated by the above algorithm, and let Ψ denote the optimum balanced partition. Then* $\text{cost}(\Psi') \leq 4 \cdot \text{cost}(\Psi)$.

Proof We begin by modifying the optimum balanced partition Ψ into a form that will be easier to analyze. Let $\{U_1, \ldots, U_k\}$ denote the subsets of Ψ, and for $1 \leq i \leq k$, let T_i denote the minimum spanning trees of the induced subgraph of U_i within the metric closure $\widehat{G}$. By standard results on Steiner trees, it follows that $\text{cost}(T_i) \leq 2 \cdot \text{cost}(U_i)$. Henceforth, we will measure costs in terms of the minimum spanning trees over the components. In particular, define $\text{cost}'(\Psi) = \sum_i \text{cost}(T_i)$. We have $\text{cost}'(\Psi) \leq 2 \cdot \text{cost}(\Psi)$. Because the cost of the minimum Steiner tree does not exceed the cost of the minimum spanning tree, we have $\text{cost}(\Psi') \leq \text{cost}'(\Psi')$.

For each pair of components of $U_i, U_j \in \Psi$, let w_i and w_j denote the maximum weights of any edges of T_i and T_j, respectively. If there exists an edge of $\widehat{G}$ that connects two vertices, one in U_i and one in U_j, such that the weight of this edge is not greater than $\min(w_i, w_j)$, then merge the vertices of U_i and U_j into a single component. Repeat this process until there is no inter-component edge that satisfies this property. Clearly, the resulting set of components, which we denote by Ψ'', is a balanced partition.

We claim that $\text{cost}'(\Psi'') \leq 2 \cdot \text{cost}'(\Psi)$. This follows from a simple charging argument. Let us consider the smallest weight edge that satisfies the merging condition. Each of the two components U_i and U_j that are connected by this edge each have edges with weights w_i and w_j, respectively, both of which are as large as the connecting edge. We add this connecting edge and charge it to the smaller of w_i and w_j. The other edge remains to be charged for future merges. In this manner, every time two components are merged, an edge from one of the components pays for the newly added connecting edge, and the other remains for future charges. Therefore, only the original edges are charged, and no edge is charged more than once. It follows that $\text{cost}'(\Psi'') \leq 2 \cdot \text{cost}'(\Psi)$, as desired.

We assert that Ψ' is a refinement of Ψ'', meaning that each set U_i' of Ψ' is a (not necessarily proper) subset of some set U_j'' of Ψ''. We prove this by contradiction. Suppose that there was an edge (u, v) such that u and v are in the same subset Ψ', but they are in different components of Ψ''. Let us assume that (u, v) is a minimum weight edge with this property. By definition of Ψ'', the weight of (u, v) is strictly smaller than the weights of all the edges of either U_i'' or U_j'' (possibly both). Assume without loss of generality that U_i'' satisfies this property. It follows that at the time that edge (u, v) is considered by our algorithm, all the edges of $MST(U_i'')$ have been considered, which implies that our algorithm has discovered all the edges of this spanning tree. Since (u, v) is the smallest inter-component edge, there can be no other vertices that are part of this component. Since Ψ'' is a balanced partition, the sum of the vertices in this component sum to zero modulo λ, which implies that this component would have been labeled as *finished* by our algorithm, contradicting the hypothesis that the edge (u, v) was considered for insertion by our algorithm.

By the above assertion, each component of Ψ' is a subset of some component of Ψ'', implying that $\text{cost}'(\Psi') \leq \text{cost}'(\Psi'')$. In conclusion, we have

$$\text{cost}(\Psi') \;\leq\; \text{cost}'(\Psi') \;\leq\; \text{cost}'(\Psi'') \;\leq\; 2 \cdot \text{cost}'(\Psi) \;\leq\; 4 \cdot \text{cost}(\Psi),$$

as desired. □

Combining Lemmas 6.2 and 6.3(ii), it follows that our algorithm achieves an approximation factor of $4(\lambda - 1)$. While obtaining the best running time has not been a focus of this work, it is easy to see that this procedure runs in polynomial time. Let $n = |V|$. The graph $\widehat{G}$ can be computed in $O(n^3)$ time by the Floyd–Warshall algorithm [3]. The Kruskal-like algorithm for computing the balanced partition can be performed in $O(n^2 \log n)$ time, as can the algorithm of Lemma 6.2. Thus, the overall running time is $O(n^3)$.

Theorem 6.1 *Given an instance $G = (V, E)$ of the λ-CMD problem for $\lambda \geq 2$, it is possible to compute a $4(\lambda - 1)$-approximation in time $O(n^3)$, where $n = |V|$.*

7 Conclusion

In this paper we have introduced a variant of the well-known circulation problem, referred to as the λ-*CMD problem*. Given a directed graph G and vertex demands $d \in \mathbb{Z} \pmod{\lambda}$, the λ-CMD problem is that of computing a minimum-cost flow that satisfies the modular demands of every vertex. A modular demand d is satisfied by an inbound flow of d or an outbound flow of $\lambda - d$.

We have shown that 2-CMD can be solved exactly in polynomial time as, thanks to the nature of modular demands, edge direction is irrelevant and finding a minimum-cost satisfying flow is equivalent to finding a minimum-cost perfect matching between vertices (see Sect. 4 for details).

Additionally, by way of a reduction from positive 1-in-3-SAT, we showed that λ-CMD is NP-hard for $\lambda \geq 3$. This reduction is first illustrated with an example of a

reduction to 3-CMD (see Sect. 5) and later generalized for larger values of λ through a simple extension of the gadgets used in the $\lambda = 3$ example (see "Appendix 1").

Our final result is a polynomial time $4(\lambda - 1)$-approximation to λ-CMD. This approximation employs a Kruskal-like algorithm to build balanced neighborhoods that can satisfy their demands without costly external flows. To do so, however, requires that the edge directions are ignored, which induces a λ factor penalty to the approximation (see Sect. 6).

In future work we hope to do away with this limitation and decrease the approximation factor, perhaps removing the λ factor entirely. Additionally, the reduction used in the hardness proof of λ-CMD is for generalized directed graphs, meaning that these graphs are not necessarily planar. While studies of actual road networks have shown that crossings are not uncommon [8], in many urban road networks the fraction of crossings to total intersections is very small [2]. It would therefore be of interest to know whether the λ-CMD problem is hard even for planar graphs. Although there are planar variants of satisfiability, which make use of cross-over gadgets to eliminate non-planar elements (see, e.g., [13]), we have been unable to discover such a gadget in the λ-CMD context. We pose as an open problem whether planar λ-CMD is NP-hard.

Finally, we plan to extend this work by generalizing it to more robust models of traffic flow, including but not limited to multi-lane roads, bidirectional streets, mixed blocks with differing numbers of sides (particularly a mix of even- and odd-numbered sides as this disrupts the periodic patterns), and vehicles that exhibit more complex behaviors such as turning at intersections, changing lanes, etc.

Appendix A: NP-hardness Generalization

Fundamental blocks are constructed in the same manner as described in the $\lambda = 3$ example in Sect. 5.1, but for completeness, we describe them here in their general form. A fundamental block begins with λ 1-vertices, connected as a star graph (i.e., a central vertex connected to each of the remaining vertices) with edges directed outward. Each 1-vertex is also connected to a $(\lambda - 1)$-vertex via an incoming edge. Finally, these $(\lambda - 1)$-vertices each have an outgoing interface edge (Fig. 8).

The expansion module, too, remains largely unchanged. It consists of a central fundamental block with λ interface edges. A fundamental block is then connected to all but one of these edges, with the remaining edge reserved for connecting to the existing variable gadget. For each expansion module added to a variable gadget, $(\lambda-1)^2-1$ interface edges are added (one interface edge is consumed when connecting to the variable gadget, hence the -1). See Fig. 9. Just as before, we must adjust the weights of a small subset of edges so that **True** and **False** assignments incur the same cost. To do so, arbitrarily select a fundamental block that is not the hub (i.e., not the fundamental block which connects to the existing variable gadget) and set the outgoing edges from its central 1-vertex to a weight of $\frac{2\lambda-3}{\lambda-1}$.

Recall that the variable gadget described in Sect. 5.1 creates $c(v_i)\lambda$ interface edges. In truth, the fundamental block creates λ interface edges while each of the $c(v_i) - 1$ expansion modules add $(\lambda - 1)^2 - 1$ interface edges. When $\lambda = 3$, these formulae

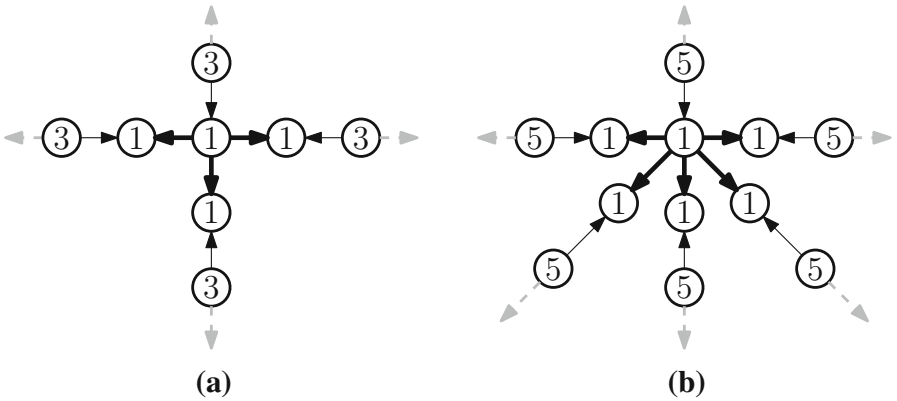

Fig. 8 Fundamental blocks with **a** $\lambda = 4$ and **b** $\lambda = 6$

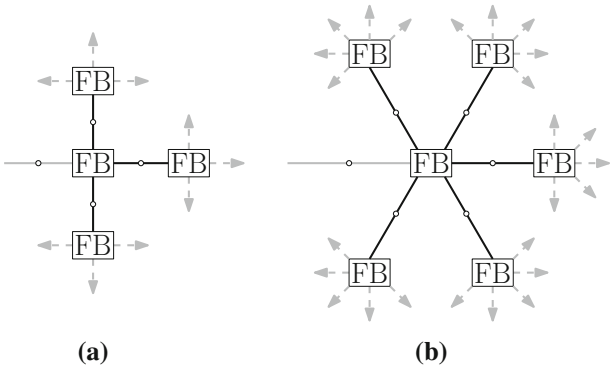

Fig. 9 Expansion modules built from fundamental blocks (FB) with **a** $\lambda = 4$ and **b** $\lambda = 6$. Edges between fundamental blocks are represented with a disc and no direction as a reminder that the blocks are connected via a shared $(\lambda - 1)$-vertex

are equivalent, each variable has $c(v_i)\lambda$ interface edges, and each of the $c(v_i)$ clause gadgets will connect to λ interface edges.

When $\lambda > 3$, the variable gadgets will have $\lambda + [c(v_i) - 1]\left[(\lambda - 1)^2 - 1\right]$ interface edges. As this is not evenly divisible by $c(v_i)$, connecting to the clause gadgets becomes problematic. To correct for this, a modified expansion module, called a *seed module*, is used to start each variable gadget rather than a fundamental block. Rather than connecting to an existing variable gadget, the edge that was previously reserved for this connection is instead attached to one of the existing interface edges (recall that in actuality a $(\lambda - 1)$-vertex is being shared, not that two edges are being connected directly). By doing so, we now have a seed module that has $(\lambda-1)^2-1$ interface edges (Fig. 10) and, when used in conjunction with unaltered expansion modules, creates a variable gadget with $c(v_i)\left[(\lambda - 1)^2 - 1\right]$ interface edges.

Clause gadgets are constructed as described in Sect. 5.2, with the caveat that they require $(\lambda - 1)^2 - 1$ vertices rather than λ. Each vertex will still only have three incoming edges, as each clause will only ever have three literals, and a demand of 1.

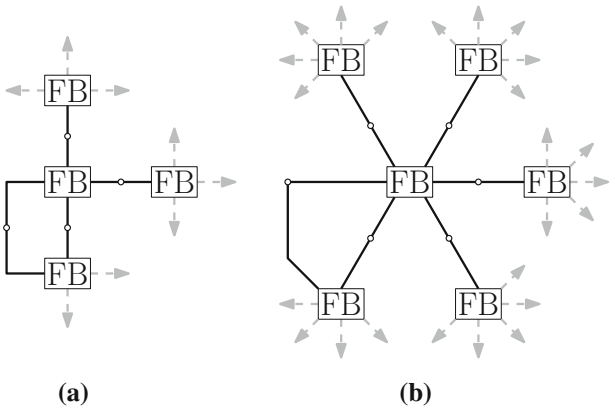

Fig. 10 Seed modules for creating variable gadgets with **a** $\lambda = 4$ and **b** $\lambda = 6$

Despite a small increase in the size of each gadget, the reduction can still be done in polynomial time.

Finally, the cost threshold must be generalized in Lemma 5.4. In the general form, if F is satisfiable there will be a satisfying flow with

$$\text{cost}(f) \leq \sum_{v_i \in V} [(\lambda^2 - 1)c(v_i) + \lambda |C| \gamma]$$

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Upper Saddle River (1993)
2. Boeing, G.: Planarity and street network representation in urban form analysis (2018). arXiv:1806.01805
3. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, vol. 2, 2nd edn. McGraw-Hill Higher Education, Boston (2001)
4. Dasler, P., Mount, D. M.: On the complexity of an unregulated traffic crossing. In: Proceedings of 14th International Symposium Algorithms Data Structure, volume 9214 of Lecture Notes Computer Science, pp. 224–235. Springer (2015)
5. Dresner, K.M., Stone, P.: A multiagent approach to autonomous intersection management. J. Artif. Int. Res. **31**, 591–656 (2008)
6. Edmonds, J., Johnson, E.L.: Matching, Euler tours and the Chinese postman. Math. Program. **5**, 88–124 (1973)
7. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM **19**, 248–264 (1972)
8. Eppstein, D., Gupta, S.: Crossing patterns in nonplanar road networks (2017). arXiv:1709.06113
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
10. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by canceling negative cycles. J. ACM **36**, 873–886 (1989)
11. Guler, S.I., Menendez, M., Meier, L.: Using connected vehicle technology to improve the efficiency of intersections. Transp. Res. Part C Emerg. Technol. **46**, 121–131 (2014)
12. Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley, Boston (2005)
13. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. **11**, 329–343 (1982)

14. Orlin, J.B.: A polynomial time primal network simplex algorithm for minimum cost flows. Math. Program. **78**, 109–129 (1997)
15. Tachet, R., Santi, P., Sobolevsky, S., Reyes-Castro, L.I., Frazzoli, E., Helbing, D., Ratti, C.: Revisiting street intersections using slot-based systems. PLoS One **11**(3), e0149607 (2016)
16. Vial, J. J. B., Devanny, W. E., Eppstein, D., Goodrich, M. T.: Scheduling autonomous vehicle platoons through an unregulated intersection. In: Goerigk, M., & Werneck, R. (eds.) 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016), vol. 54 of OpenAccess Series in Informatics (OASIcs), pp. 1–14. Schloss DagstuhlLeibniz-Zentrum fuer Informatik (2016)
17. Yu, J., LaValle, S. M.: Multi-agent path planning and network flow. In: Frazzoli, E., Lozano-Perez, T., Roy, N., & Rus, D. (eds.) Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics, pp. 157–173, Springer, Berlin, Heidelberg (2013)