Modular Circulation and Applications to Traffic Management[⋆]

Philip Dasler and David M. Mount

Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland, College Park, Maryland 20742
{daslerpc,mount}@cs.umd.edu

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 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

[⋆] Research supported by NSF grant CCF-1618866.

© Springer International Publishing AG 2017

from $\mathbb{Z}_\lambda$ to the edges of G. For each vertex $v \in V$, define

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

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, 2, 6, 8]. 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 Section 4).
- 3-CMD is NP-hard (see Section 5).
- There is a polynomial time $4(\lambda-1)$-approximation to λ-CMD (see Section 6).

In Section 2 we discuss the relevance of the λ-CMD problem to a traffic-management problem. In Section 3 we present some preliminary observations regarding this problem. In Sections 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 [9]. There are numerous approaches to modeling traffic flow and diverse computational approaches to solve and analyze the associated traffic management problems [4,11]. 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 [3], we considered the problem of scheduling the movements of a collections of vehicles through a system of unregulated crossing. 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 motion at the motion of these groups [7, 10].

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 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. 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 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. 1(a)). 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. 1(b)).

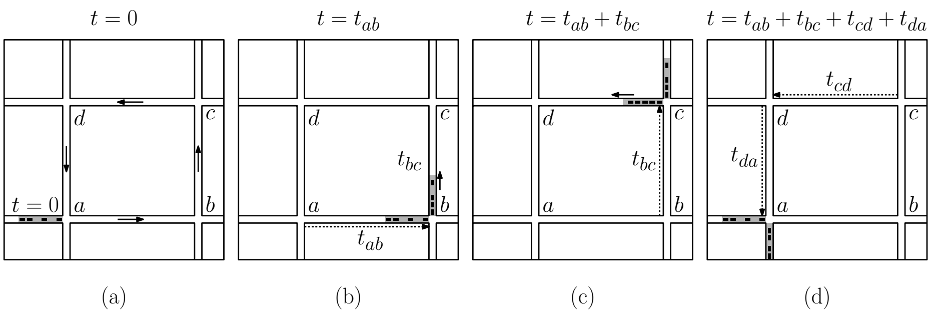

Fig. 1. Delay-free traffic-light schedule.

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. 1(c)). On returning to a (see Fig. 1(d)), 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

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

Since the transportation times along the road segments are not under our control, in order to satisfy this constraint, we introduce an (ideally small) 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. Due to space limitations, the proofs of this and many other lemmas have been omitted and will appear in the full version of the paper.

Lemma 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.*

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 2. *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 $d(V) \equiv 0 \pmod{\lambda}$.*

It follows from this lemma 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 separately on each connected component of G'. 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 2(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 $wt(\pi(u, v))$. (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 3. *lem:2-cmd 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 $\widehat{G}$.*

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 [2]. Thus, the overall running time is $O(n^3)$.

Theorem 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 3-CMD

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

Theorem 2. *For $\lambda \geq 3$, the λ-CMD problem is NP-hard.*

The reduction is from positive 1-in-3-SAT [5]. For the sake of brevity and simplicity, we show the proof for the case $\lambda = 3$ here, but the method easily generalizes (as will be explained in the full version of the paper). 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 Figure 2(a). 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 Figure 2(b)). 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 Figure 2(c)).

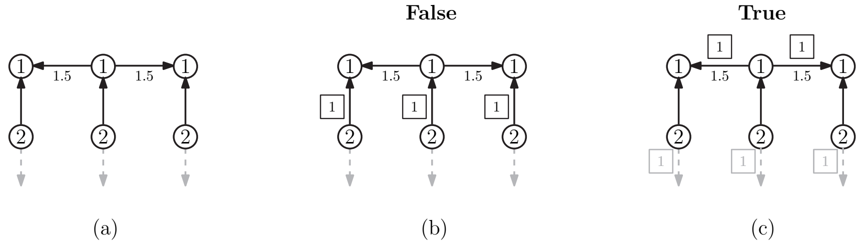

Fig. 2. (a) The fundamental building block used to build variable gadgets. Interface edges are dashed gray segments. (b,c) CMDs representing the assignment of **False** and **True** values, respectively, with the flow values in boxes.

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 4. *Given a fundamental block, a satisfying flow has cost ≤ 3 if and only if that flow is 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 2(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 (mod λ) (see Lemma 2). 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 over all 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 Figure 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.

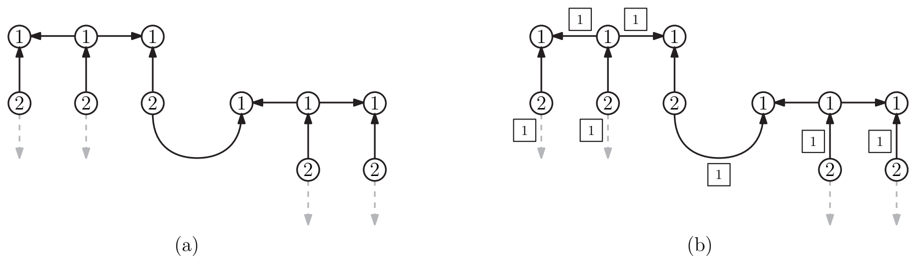

Fig. 3. Two fundamental blocks connected via a shared 2-vertex, with figure (b) showing a flow that is satisfying but not interface-consistent.

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 Figure 4). When attached to a variable, this module creates four new interface edges and consumes one, thus extending the variable by three interface edges.

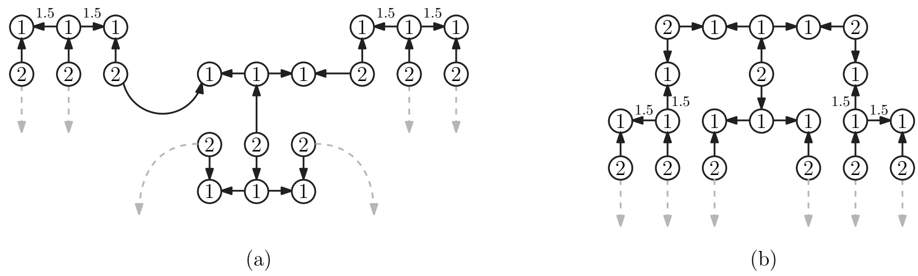

Fig. 4. (a) A fundamental block with a single extension module attached. (b) The same structure rearranged to emphasize the two clause outputs, each with three interface edges.

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 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 4, and the fact that the expansion module is constructed from fundamental blocks, we have the following:

Lemma 5. *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. Doing so provides $c(v_i)\lambda$ interface edges and yields the following result:

Lemma 6. *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 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 Figure 5). 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).

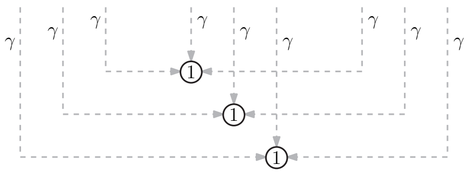

Fig. 5. A full clause gadget, with three inputs from each of three literals.

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. 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.

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.

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 4, these inconsistencies will always lead to a strictly greater cost. Thus:

Lemma 7. *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 an $4(\lambda - 1)$-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 Section 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 Section 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 8. *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 2(ii), we may assume that $d(V) \equiv 0 \pmod{\lambda}$, and so there is always a trivial partition, namely $\{V\}$ itself. Define $\mathrm{cost}(U_i)$ to be $\mathrm{cost}(SMT(U_i))$, 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 9. *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* $\mathrm{cost}(\Psi) \leq |f| \leq (\lambda - 1) \cdot \mathrm{cost}(\Psi)$.

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

Combining Lemmas 9 and 10(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 [2]. 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 9. Thus, the overall running time is $O(n^3)$.

Theorem 3. *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 $n = |V|$.*

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, 1993.
2. T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.
3. P. Dasler and D. M. Mount. On the complexity of an unregulated traffic crossing. In *Proc. 14th Internat. Sympos. Algorithms Data Struct.*, volume 9214 of *Lecture Notes Comput. Sci.*, pages 224–235. Springer-Verlag, 2015.
4. K. M. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *J. Artif. Int. Res.*, 31:591–656, 2008.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.
6. A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36:873–886, 1989.
7. S. I. Guler, M. Menendez, and L. Meier. Using connected vehicle technology to improve the efficiency of intersections. *Transportation Research Part C: Emerging Technologies*, 46:121–131, 2014.
8. J. B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78:109–129, 1997.
9. R. Tachet, P. Santi, S. Sobolevsky, L. I. Reyes-Castro, E. Frazzoli, D. Helbing, and C. Ratti. Revisiting street intersections using slot-based systems. *PLOS ONE*, 11(3):e0149607, 2016.
10. J. J. B. Vial, W. E. Devanny, D. Eppstein, and M. T. Goodrich. Scheduling autonomous vehicle platoons through an unregulated intersection. In M. Goerigk and R. Werneck, editors, *16th Wkshp. Alg. Approaches Transport. Model., Opt., and Syst. (ATMOS 2016)*, volume 54, pages 1–14, 2016.
11. J. Yu and S. M. LaValle. Multi-agent path planning and network flow. In E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, editors, *Algorithmic Foundations of Robotics X*, pages 157–173, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.