2015

Operator Calculus Algorithms for Multi-Constrained Paths

Jamila Ben Slimane
MEDIATRON - SupCom Tunis, Tunisia, jamila.benslimane@isetcom.tn

Rene' Schott
LORIA - Universite' de Lorraine, schott@loria.fr

Ye Qiong Song
LORIA - Universite' de Lorraine, ye-qiong.song@loria.fr

G. Stacey Staples
Southern Illinois University Edwardsville, sstaple@siue.edu

Evangelia Tsiontsiou
LORIA - Universite' de Lorraine, vtsiontsiou@gmail.com

International Journal of Mathematics and
Computer Science, **10**(2015), no. 1, 69–104

Operator Calculus Algorithms for Multi-Constrained Paths

Jamila Ben Slimane[1], René Schott[2], Ye-Qiong Song[2], G. Stacey Staples[3], Evangelia Tsiontsiou[2]

[1]MEDIATRON - SupCom
Tunis, Tunisia

[2]LORIA - Université de Lorraine
Campus Scientifique, B.P. 239, 54506
Vandoeuvre-lès-Nancy, France

[3]Department of Mathematics and Statistics
Southern Illinois University Edwardsville
Edwardsville, IL 62026-1653, USA

email: jamila.benslimane@isetcom.tn, rene.schott@loria.fr,
ye-qiong.song@loria.fr, sstaple@siue.edu, vtsiontsiou@gmail.com

(Received February 28, 2015, Accepted April 3, 2015)

Abstract

Classical approaches to multi-constrained routing problems generally require construction of trees and the use of heuristics to prevent combinatorial explosion. Introduced here is the notion of constrained path algebras and their application to multi-constrained path problems. The inherent combinatorial properties of these algebras make them useful for routing problems by implicitly pruning the underlying tree structures. Operator calculus (OC) methods are generalized to multiple non-additive constraints in order to develop algorithms for the multi constrained path problem and multi constrained optimization problem. Theoretical underpinnings are developed first, then algorithms are presented. These algorithms demonstrate the tremendous simplicity, flexibility and speed of the OC approach. Algorithms

Key words and phrases: Routing problems, shortest path, quality of service, zeons, nilpotent adjacency matrix.
AMS (MOS) Subject Classifications: 05C30, 05E15, 90C35.
ISSN 1814-0432, 2015, http://ijmcs.future-in-tech.net

are implemented in *Mathematica* and *Java* and applied to a problem first proposed by Ben Slimane *et al.* as an example.

1 Introduction

The objective of Quality of Service (QoS) routing is to find a path from a source to a destination using the minimum amount of network resources such as energy consumption per route, residual battery power per node, available time slots, etc. while satisfying various QoS constraints, such as delay, reliability, etc.

When multiple routing metrics are considered, the problem becomes a multi-constrained path problem, which has been mathematically proven to be NP-complete [22]. In practice, however, these problems are often solved in polynomial time. In fact, van Mieghem and Kuipers [21] suggest that QoS routing in realistic networks may not be NP-complete. More specifically, they suggest that QoS routing problems confined to a particular class of networks (topology and link weight structure) may not be NP-complete.

Many heuristic and approximation algorithms have been developed to identify paths given particular QoS requests. For example, in [23], Xue, *et al.* study the NP-hard multi-constrained QoS routing problem of seeking a path from a source to a destination in the presence of multiple additive end-to-end QoS constraints. By considering an optimization version of the problem in which the first constraint is enforced and the remaining constraints are approximated, they obtain $(1 + \epsilon)$-approximations for any ϵ.

In [8], Hou, *et al.* study an NP-complete routing problem with multiple additive constraints, and develop an approximation algorithm using a novel approach they call the "two-dimensional sampling scheme." This technique reduces the approximation error without increasing the computational complexity.

While classical approaches to multi-constrained routing problems generally require construction of trees and the use of heuristics to prevent combinatorial explosion [7, 11], the operator calculus approach presented herein allows such explicit tree constructions to be avoided. Introduced here is the notion of constrained path algebras and their application to multi-constrained path problems. The inherent combinatorial properties of these algebras make them useful for routing problems by implicitly pruning the underlying tree structures.

Algebraic approaches to QoS routing in the internet have been studied for some time. In his algebraic approach of 2002, Sobrinho [17] defined an

algebra of weights containing a binary operation for the composition of link weights into path weights and an order relation. Sobrinho further showed that for a generalized Dijkstra's algorithm to yield optimal paths, it is both necessary and sufficient for the algebra to have a property called *isotonicity*, in which the order relation between the weights of any two paths is preserved if both are prefixed or appended by a common third path.

Operator calculus (OC) methods on graphs have been developed in a number of works by Schott and Staples [15, 16, 18, 19]. The principal idea underlying the approach is the association of graphs with algebraic structures whose properties reveal information about the associated graphs. For example, an element a of a ring or algebra is said to be *nilpotent* if $a^n = 0$ for some positive integer n. By constructing the "nilpotent adjacency matrix" associated with a finite graph, information about self-avoiding structures (paths, cycles, trails, etc.) in the graph are revealed by computing powers of the matrix. Cycles are removed from consideration automatically by the algebra itself.

At the heart of the OC approach are "zeon" algebras, which can be considered a commutative version of the algebra of fermion creation (or annihilation) operators well-known to quantum physicists. Nilpotent adjacency matrices can be regarded as quantum random variables in an algebraic probability space [16]. The OC algorithms presented here are most naturally expressed in the language of operator theory using Dirac notation. Generalizations of zeon algebras provide tools for sieving out paths with multidimensional weights (or costs) simultaneously satisfying a number of constraints [14]. Paths whose weights exceed any of the constraints are zeroed out by the algebra's nilpotent properties.

In this paper, the impact of the OC approach is further extended by developing a "constraints algebra" that automatically removes from consideration any path whose weight fails to simultaneously satisfy multiple constraints. The resulting constrained path algebra is then used to devise algorithms for computing multi-constrained paths in weighted graphs.

Unlike Sobrinho's approach, the OC approach does not involve Dijkstra's algorithm, and the algebras involved are all derived from algebras already known from quantum physics. The OC approach is well suited for implementation in a computer algebra system, e.g. *Mathematica*, and can be approached with tools of operator theory and linear algebra. Unlike algorithms relying on Dijkstra's algorithm (e.g., SAMCRA [10] and Sobrinho's algorithm), the OC approach works with negative edge weights. Further, the OC approach simultaneously finds all feasible multi-constrained paths (on

the minimum number of hops) between a source and a target.

Consider a directed graph $G = (V, E)$ on n vertices such that associated with each edge $(v_i, v_j) \in E$ is a vector weight $\mathbf{w}_{ij} = (w_{ij1}, \ldots, w_{ijm}) \in \mathbb{R}^m$. The point $w^* = (w_1^*, \ldots, w_m^*) \in X \subset \mathbb{R}^m$ is referred to as a *Pareto minimum* of X if there does not exist $\mathbf{w} = (w_1, \ldots, w_m) \in X$ such that

$$(\forall i)\,[w_i \leq w_i^*], \text{ and} \tag{1.1}$$

$$(\exists j)\,[w_j < w_j^*]. \tag{1.2}$$

Equivalently, one says that w^* is *nondominated from below*.

Defining the weight of a path in an edge-weighted graph as the sum of vector weights of arcs contained in the path, a *Pareto path* is then a path whose weight is a Pareto minimum.

For the case $m = 1$, Dijkstra's algorithm finds all single source minimum paths in a directed graph on n vertices with nonnegative edge weights in $\mathcal{O}(n^2)$ time [5]. The Bellman-Ford algorithm finds single source minimal paths in digraphs with arbitrary edge weights and runs in $\mathcal{O}(n\,|E|)$ time [1, 6].

In the more general case $m \geq 1$, Corley and Moon [3] presented an algorithm for finding all Pareto paths with complexity $\mathcal{O}(mn^{2n-3} + mn^n)$.

The aim of the current work is to find generalized Pareto paths satisfying multiple constraints involving weights that are not necessarily additive. In place of vector addition, m-weights will be allowed arbitrary associative binary operations assigned componentwise. In place of the ordinary real relational operators $\leq$ and $<$, more general transitive relations will be considered.

2 Operator Calculus Approach

A simple example of the task at hand is to compute all feasible hop-minimal paths from v_1 to v_5 in the five-node graph of Figure 1. Each edge is weighted with a pair of additive weights subject to the constraints vector $(0.5, 100)$, so that a path $\mathbf{p}$ of multiweight (w_1, w_2) is feasible only if $w_1 \leq 0.5$ and $w_2 \leq 100$. This problem will be solved in Example 2.18 using algebraic tools developed below.

Definition 2.1. *Let $m \in \mathbb{N}$, and let $\mathfrak{C} \in \mathbb{R}^m$. Let $*$ denote an m-tuple of associative binary operations on $\mathbb{R}$. Let $\mathfrak{R} = (R_1, \ldots, R_m)$ denote an m-tuple*

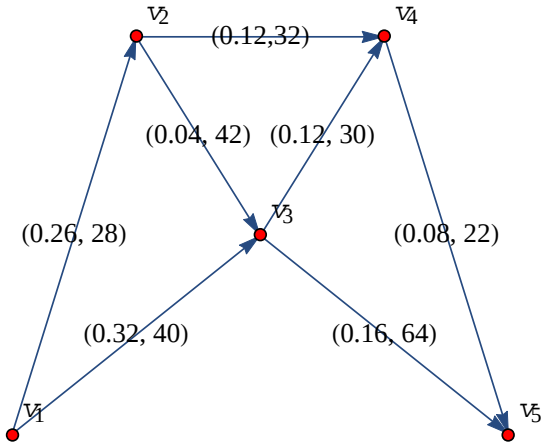

Figure 1: Five node multi-weighted graph.

of reflexive, transitive, antisymmetric relations on $\mathbb{R}$*, and note that* $\mathfrak{R}$ *itself is a transitive relation on* $\mathbb{R}^m$*. The vector* $\mathfrak{C}$ *is called the* constraint vector *relative to the* constraints relation $\mathfrak{R}$*. A vector* $\mathbf{x} \in \mathbb{R}^m$ *is said to* satisfy $\mathfrak{C}$*, denoted* $\mathbf{x} \preceq \mathfrak{C}$*, if and only if*

$$(\mathbf{x}, \mathfrak{C}) \in \mathfrak{R}.$$

In particular, $\mathbf{x} \preceq \mathfrak{C}$ *if and only if* $(x_i, c_i) \in R_i$ *for all* $i = 1, \ldots, m$*. Equivalently, one writes*

$$\mathbf{x} \preceq \mathfrak{C} \Leftrightarrow x_i \preceq_i c_i \ \forall i \in \{1, \ldots, m\}.$$

Note that for each $i = 1, \ldots, m$*, the relation* R_i *determines a partial order on* $\mathbb{R}$*; i.e., for* $1 \leq i \leq m$*,*

$$(x_i \preceq_i y_i) \wedge (y_i \preceq_i x_i) \Rightarrow x_i = y_i.$$

Given a finite graph G in which each edge is weighted with an m-tuple of nonnegative integers and a constraint vector $\mathfrak{C} = (c_1, \ldots, c_m) \in \mathbb{R}^m$, the multi-constrained path problem is defined as follows.

Definition 2.2. *The* Multi-Constrained Path problem *(or MCP problem) is to find paths* $\mathbf{p}$ *from source vertex* v_0 *to target vertex* v_∞ *in the graph* G *such that*

$$\mathrm{wt}(\mathbf{p}) \preceq (c_1, \ldots, c_m) = \mathfrak{C}.$$

Given an initial vertex v_0 and terminal vertex v_∞ in a weighted graph, the collection of *feasible paths* from v_0 to v_∞ refers to all paths whose associated total costs satisfy some predefined constraints. Among these feasible paths, an optimal or preferred path can then be chosen.

For convenience, $\mathcal{P}$ will denote the set of *all paths* in the graph, G, and $\mathfrak{P}$ will denote the set of *all feasible paths* in G, i.e.,

$$\mathfrak{P} := \{\mathbf{p} \in \mathcal{P} : \mathrm{wt}(\mathbf{p}) \preceq \mathfrak{C}\}.$$

An important variant of the MCP problem that is of particular interest is the associated optimization problem.

Definition 2.3. *Letting v_0 and v_∞ denote fixed source and target vertices in a multi-weighted graph G, the* Multi-Constrained Optimal Path problem *(or MCOP problem) is to find a path $\mathbf{p} = (v_0, \ldots, v_\infty) \in \mathcal{P}$ such that*

$$\mathrm{wt}(\mathbf{p}) \preceq \mathrm{wt}(\mathbf{q}) \ \forall \mathbf{q} \in \mathfrak{P}.$$

Given a constraint vector $\mathfrak{C} = (c_1, \ldots, c_m) \in \mathbb{R}^m$, a path $\mathbf{p}$ is deemed *feasible* if its vector weight satisfies

$$\mathrm{wt}(\mathbf{p}) = (w_1, \ldots, w_m) \preceq \mathfrak{C}.$$

Let $\mathfrak{P}$ denote the collection of all feasible paths in G. Fixing source vertex v_0 and target vertex v_∞, the goal is to find a path in $\mathbf{p} = (v_0, \ldots, v_\infty) \in \mathfrak{P}$ whose weight is a Pareto minimum. The operator calculus approach described herein can be applied to sieve out the collection of feasible paths and recover all single-source Pareto paths remaining.

Under the assumption that G has no multiple edges, paths are uniquely determined by vertex sequences; e.g., $\mathbf{p} = (p_0, \ldots, p_\ell)$. Let $\mathbf{w}_i$ denote the multi-weight of the i^{th} edge in the path $\mathbf{p}$. The *weight* of $\mathbf{p}$ is then defined by

$$\mathrm{wt}(\mathbf{p}) := \mathbf{w}_1 * \mathbf{w}_2 * \cdots * \mathbf{w}_\ell.$$

Definition 2.4. *The* constraints algebra*, denoted $\mathcal{A}_\mathfrak{C}$, is the real associative unital algebra generated by $\{\xi^\mathbf{x} : \mathbf{x} \in \mathbb{R}^m\}$ with (formal) unit $\xi^\mathbf{0}$ having multiplication defined according to*

$$\xi^\mathbf{x}\, \xi^\mathbf{y} := \begin{cases} \xi^{\mathbf{x}*\mathbf{y}} & \text{if } \mathbf{x} * \mathbf{y} \preceq \mathfrak{C}, \\ 0 & \text{otherwise.} \end{cases}$$

Example 2.5. *Consider* $m = 3$, $* = (+, \cdot, \max)$, $\mathfrak{C} = (10, 20, 50)$, *and* $\mathfrak{R} = (\leq, \leq, \geq)$. *In this case,*

$$\xi^{(1,4,10)}\,\xi^{(9,3,80)} = \xi^{(1+9,4\cdot 3,\max\{10,80\})} = \xi^{(10,12,80)}.$$

On the other hand, because $1 + 12 > 10$ *and also because* $\max\{30, 35\} < 50$, *one finds*

$$\xi^{(1,4,30)}\,\xi^{(12,3,35)} = 0.$$

In this example, the multiplicative identity (unit) of $\mathcal{A}_{(10,20,50)}$ *is formally defined as* $\xi^{\mathbf{0}} := \xi^{(0,1,-\infty)}$.

Given a constraint vector $\mathfrak{C} = (c_1, \ldots, c_m) \in \mathbb{R}^m$, properties of the constraints algebra $\mathcal{A}_{\mathfrak{C}}$ can be used to sieve out the feasible paths from the collection of all paths. The feasible paths can then be ranked and an optimal path chosen.

For fixed positive integer n, consider alphabet $\Sigma_n := \{\omega_i : 1 \leq i \leq n\}$. For convenience, we adopt the following *ordered multi-index* notation. In particular, letting $\mathbf{u} = (u_1, \ldots, u_k)$ for some k, the notation $\omega_{\mathbf{u}}$ will be used to denote a *sequence* (or *word*) of distinct symbols of the form

$$\omega_{\mathbf{u}} := \omega_{u_1}\omega_{u_2}\cdots\omega_{u_k}.$$

Appending 0 to the set Σ_n, multiplication is defined on the words constructed from elements of Σ_n by

$$\omega_{\mathbf{u}}\omega_{\mathbf{v}} = \begin{cases} \omega_{\mathbf{u}\,.\mathbf{v}} & \text{if } \mathbf{u} \cap \mathbf{v} = \emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathbf{u}\,.\mathbf{v}$ denotes sequence concatenation.

One thereby obtains the noncommutative semigroup Ω_n, whose elements are the symbol 0 along with all *finite words* on distinct generators (i.e., finite sequences of distinct symbols from the alphabet Σ_n). Since there are only n generators, it is clear that the maximum multi-index size of semigroup elements is n. Moreover, these symbols can appear in any order so that the order of the semigroup is $\displaystyle\sum_{k=0}^{n} \binom{n}{k} k! = \sum_{k=0}^{n} (n)_k$. Here, $(n)_k$ denotes the falling factorial.

Defining (vector) addition and real scalar multiplication on the semigroup yields the semigroup algebra $\mathbb{R}\Omega_n$ of dimension $|\Omega_n|$. This semigroup algebra will be referred to as a *path algebra.* The next step is defining a nilpotent adjacency matrix that preserves path-identifying information.

Definition 2.6. *Let $G = (V, E)$ be a graph on n vertices, either simple or directed with no multiple edges. Let $\{\omega_i\}$, $1 \leq i \leq n$ denote the null-square, noncommutative generators of $\mathbb{R}\Omega_n$. Define the* path-identifying nilpotent adjacency matrix Ξ *associated with G as the $n \times n$ matrix*

$$\Xi_{ij} = \begin{cases} \omega_j \text{ if } (v_i, v_j) \in E, \\ 0 \text{ otherwise.} \end{cases} \tag{2.3}$$

Given a labeling of V by the n-set, $[n] := \{1, \ldots, n\}$, and recalling Dirac notation, the i^{th} row of Ξ is conveniently denoted by $\langle v_i | \, \Xi$ while the j^{th} column is denoted by $\Xi \, | v_j \rangle$. In this way, Ξ is completely determined by

$$\langle v_i | \Xi | v_j \rangle = \begin{cases} \omega_j & \text{if there exists a directed edge } v_i \to v_j \text{ in } G, \\ 0 & \text{otherwise,} \end{cases}$$

for all vertex pairs $(v_i, v_j) \in E$.

Theorem 2.7. *Let Ξ be the path-identifying nilpotent adjacency matrix of an n-vertex graph G. For any $k > 1$ and $1 \leq i \neq j \leq n$,*

$$\omega_i \left\langle v_i | \Xi^k | v_j \right\rangle = \sum_{k\text{-paths } \mathbf{w}: v_i \to v_j} \omega_{\mathbf{w}}.$$

Moreover,

$$\left\langle v_i | \Xi^k | v_i \right\rangle = \sum_{k\text{-cycles } \mathbf{w} \text{ based at } v_i} \omega_{\mathbf{w}}.$$

More specifically, when $i \neq j$, the product of ω_i with the entry in row i, column j of Ξ^k is a sum of basis blades indexed by k-step paths $v_i \to v_j$ in G. Moreover, entries along the main diagonal of Ξ^k are sums of basis blades indexed by the graph's k-cycles.

Proof. The result follows from straightforward mathematical induction on k using properties of the multiplication in $\mathbb{R}\Omega_n$ with the observation that the initial vertex of the walk, v_i, is unaccounted for in $\langle v_i | \Xi^k | v_j \rangle$, as seen in (2.3) of the matrix definition. Hence, each term of $\left\langle v_i | \Xi^k | v_j \right\rangle$ is indexed by the vertex sequence of a k-walk from v_i to v_j with no repeated vertices, except possibly v_i at some intermediate step. Left multiplication by ω_i thus sieves out the k-paths.

Considering entries along the main diagonal of Ξ^k, note that the final step of a k-cycle based at v_i returns to v_i so that left multiplication by ω_i is not required for cycle enumeration.

□

The goal now is to extend the path-identifying nilpotent adjacency matrix approach to include weighted edges. In particular, each edge of the graph will be weighted by an m-tuple of real numbers. In this manner, paths in the graph will have associated m-dimensional weights. Typically, the components of these weights are nonnegative integers or real numbers, and the weights are additive.

Definition 2.8. *Given a path algebra Ω_n, the tensor algebra $\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$ is referred to as the* constrained path algebra *relative to* $(\mathfrak{C}, \mathfrak{R})$.

Definition 2.9. *Let $\mathfrak{C} \in \mathbb{R}^m$, and let $G = (V, E)$ be a graph on n vertices whose edges (v_i, v_j) are multi-weighted by vectors $\mathbf{w}_{ij} \in {\mathbb{R}_+}^m$. The* $\mathfrak{C}$-constrained path-identifying nilpotent adjacency matrix *associated with G is the $n \times n$ matrix with entries in $\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$ determined by*

$$\Psi_{ij} = \begin{cases} \xi^{\mathbf{w}_{ij}} \omega_j \text{ if } (v_i, v_j) \in E, \\ 0 \text{ otherwise.} \end{cases}$$

The $\mathfrak{C}$-constrained path-identifying nilpotent adjacency matrix Ψ represents an algebra homomorphism via

$$\omega_{v_i} \mapsto \sum_{\ell} \langle v_i | \Psi | v_\ell \rangle.$$

This extends inductively to the full algebra $\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$ by linear extension of

$$\omega_{\mathbf{p}.v_i} \mapsto \omega_{\mathbf{p}} \sum_{\ell} \langle v_i | \Psi | v_\ell \rangle.$$

Dirac notation is extended to $(\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n)^{|V|}$ by linear extension of

$$\langle \xi^{\mathbf{a}} \omega_{\mathbf{b}} | := \xi^{\mathbf{a}} \omega_{\mathbf{b}} \langle \mathbf{b}_{|\mathbf{b}|} |.$$

Example 2.10. *For example, regarding Ψ as a matrix whose rows are indexed by the vertices $\{v_i\}$,*

$$\langle \xi^{\mathbf{a}} \omega_{(v_1,\ldots,v_k)} | \Psi = \xi^{\mathbf{a}} \omega_{(v_1,\ldots,v_k)} \langle v_k | \Psi,$$

which yields the row vector obtained by component-wise multiplication of the v_k-th row of Ψ by the fixed element $\xi^{\mathbf{a}} \omega_{(v_1,\ldots,v_k)}$.

For real scalars $\alpha, \beta \in \mathbb{R}$, linear extension is given by

$$\langle \alpha\xi^{\mathbf{a}}\omega_{\mathbf{b}} + \beta\xi^{\mathbf{c}}\omega_{\mathbf{d}}|\Psi := \alpha\xi^{\mathbf{a}}\omega_{\mathbf{b}}\langle \mathbf{b}_{|\mathbf{b}|}|\Psi + \beta\xi^{\mathbf{c}}\omega_{\mathbf{d}}\langle \mathbf{d}_{|\mathbf{d}|}|\Psi.$$

Define $|\mathbf{1}\rangle$ as the column vector of all 1s. The inner product of a vector $\mathbf{v}$ with $|\mathbf{1}\rangle$ is simply the sum of the components of $\mathbf{v}$. The mapping from $(\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n)^{|V|}$ back to $\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$ is then given naturally by the inner product with $|\mathbf{1}\rangle$.

Theorem 2.11. *Given a multi-weighted graph G on n vertices and a constraint vector $\mathfrak{C} = (c_1, \ldots, c_m)$, let v_0 and v_∞ denote distinct source and target vertices, respectively. If Ψ is the $\mathfrak{C}$-constrained path-identifying nilpotent adjacency matrix associated with G, then the collection of feasible paths $v_0 \to v_\infty$ in G is given by*

$$\xi^{\mathbf{0}}\omega_0 \sum_{\ell=1}^{n} \langle v_0|\Psi^\ell|v_\infty\rangle = \sum_{\substack{\text{paths } \mathbf{p}: v_0 \to v_\infty \\ \text{wt}(\mathbf{p}) < \mathbf{c}}} \xi^{\text{wt}(\mathbf{p})}\omega_{\mathbf{p}}.$$

More specifically, feasible paths exist if and only if $\xi^{\mathbf{0}}\omega_0 \sum_{\ell=1}^{n} \langle v_0|\Psi^\ell|v_\infty\rangle$ is nonzero. For the case $v_0 = v_\infty$, one has

$$\langle v_0|\Psi^\ell|v_0\rangle = \sum_{\substack{\text{cycles } \mathbf{p}: v_0 \to v_0 \\ \text{wt}(\mathbf{p}) < \mathbf{c}}} \xi^{\text{wt}(\mathbf{p})}\omega_{\mathbf{p}}.$$

Proof. The result follows from Theorem 2.7 in consideration of combinatorial properties of $\mathcal{A}_{\mathfrak{C}}$. □

Note that the partial order $\preceq$ on the multi-exponents appearing in the canonical expansion of any element $u \in \mathcal{A}_{\mathfrak{C}}$ can be totally ordered by sorting. Writing $\wedge$ and $\vee$ for the logical conjunction (AND) and disjunction (OR), this ordering is done lexicographically according to

$$\mathbf{x} \preccurlyeq \mathbf{y} \Leftrightarrow (x_1 \prec_1 y_1) \vee [(x_1 = y_1) \wedge (x_2 \prec y_2)] \\ \vee \cdots \vee [(\forall i < m,\, x_i = y_i) \wedge (x_m \prec_m y_m)]\,. \quad (2.4)$$

A path $\mathbf{p} \in X \subset \mathfrak{P}$ is said to be *optimal* or *preferred* for X if

$$\text{wt}(\mathbf{p}) \preccurlyeq \text{wt}(\mathbf{q}),\ (\forall \mathbf{q} \in X).$$

The ordering $\preccurlyeq$ of multi-exponents naturally induces an ordering on the monomials $\{\xi^{\text{wt}(\mathbf{p})}\omega_{\mathbf{p}} : \mathbf{p} \in \mathfrak{P}\}$. One then defines a *choice function* $\beth$ on $\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$ by

$$\beth\left(\sum_{\mathbf{p}\in X} \xi^{\mathbf{a}(\mathbf{p})}\omega_{\mathbf{p}}\right) := \{\xi^{\text{wt}}(\mathbf{p})\omega_{\mathbf{p}} : \text{wt}(\mathbf{p}) \preccurlyeq \text{wt}(\mathbf{q}),\ \forall \mathbf{q} \in X\}.$$

Corollary 2.12. *If* $\xi^{\mathbf{0}}\omega_0 \sum_{\ell=1}^{n} \langle v_0|\Psi^\ell|v_\infty\rangle \neq 0$, *then the optimal (preferred) path* $\mathbf{p} = (v_0, \ldots, v_\infty)$ *exists and is given by*

$$\beth\left(\xi^{\mathbf{0}}\omega_0 \sum_{\ell=1}^{n} \langle v_0|\Psi^\ell|v_\infty\rangle\right) = \xi^{\mathrm{wt}(\mathbf{p})}\omega_{\mathbf{p}}.$$

Proof. By Theorem 2.11, the collection of all feasible paths $v_0 \to v_\infty$ is given by $\xi^{\mathbf{0}}\omega_0 \sum_{\ell=1}^{n} \langle v_0|\Psi^\ell|v_\infty\rangle$. By the chosen ordering of paths and definition of $\beth$, the optimal path is as stated. □

Proposition 2.13. *Given a multi-weighted graph* G *on* n *vertices with nilpotent multi-weighted adjacency matrix* Ψ, *a constraint vector* $\mathfrak{C} = (c_1, \ldots, c_m)$, *and a vertex* v_0, *the collection of all feasible paths with initial vertex* v_0 *in* G *is given by*

$$\sum_{\ell=1}^{n} \langle \xi^{\mathbf{0}}\omega_{v_0}|\Psi^\ell|\mathbf{1}\rangle = \sum_{\{\mathbf{p}\in\mathfrak{P}:p_0=v_0\}} \xi^{\mathrm{wt}(\mathbf{p})}\omega_{\mathbf{p}}.$$

Proof. By Theorem 2.11, $\langle \xi^{\mathbf{0}}\omega_{v_0}|\Psi^\ell$ is a vector whose j^{th} component is an algebraic sum representing all feasible ℓ-paths $\mathbf{p}$ from v_0 to v_j in the graph G. Computing the inner product with the ones vector thereby transforms the row vector into an algebraic sum of all components. Hence, $\langle \xi^{\mathbf{0}}\omega_{v_0}|\Psi^\ell|\mathbf{1}\rangle$ is an algebraic sum representing all paths of length ℓ having initial vertex v_0. Summing over all possible path lengths then gives the stated result. □

Remark 2.14. *Note that for real parameter* t, *a generating function can be prescribed as follows.*

$$\langle \xi^{\mathbf{0}}\omega_{v_0}|e^{t\,\Psi}|\mathbf{1}\rangle = \sum_{\ell=1}^{n} \sum_{\substack{\mathbf{p}\in\mathfrak{C}:p_0=v_0 \\ |\mathbf{p}|=\ell}} \frac{t^\ell}{\ell!}\xi^{\mathrm{wt}(\mathbf{p})}\omega_{\mathbf{p}}.$$

Feasible paths of length ℓ *are then recovered by evaluating partial derivatives.*

$$\frac{\partial^\ell}{\partial t^\ell}\langle \xi^{\mathbf{0}}\omega_{v_0}|e^{t\,\Psi}|\mathbf{1}\rangle\bigg|_{t=0} = \langle \xi^{\mathbf{0}}\omega_{v_0}|\Psi^\ell|\mathbf{1}\rangle.$$

Define the functional $\langle \mathbf{1}_{\mathcal{A}_\mathfrak{C}}|$ as the row vector

$$\langle \mathbf{1}_{\mathcal{A}_\mathfrak{C}}| := (\xi^{\mathbf{0}}\omega_{v_1}, \xi^{\mathbf{0}}\omega_{v_2}, \ldots, \xi^{\mathbf{0}}\omega_{v_n}).$$

Given a path $\mathbf{p}$ of finite but indeterminate length, let p_∞ denote the terminal vertex of path $\mathbf{p}$.

Proposition 2.15. *Given a multi-weighted graph G on n vertices with nilpotent multi-weighted adjacency matrix Ψ, a constraint vector $\mathfrak{C} = (c_1, \ldots, c_m)$, and a vertex v_∞, the collection of all feasible paths with terminal vertex v_∞ in G is given by*

$$\sum_{\ell=1}^{n} \langle \mathbf{1}_{\mathcal{A}_\mathfrak{C}} | \Psi^\ell | v_\infty \rangle = \sum_{\{\mathbf{p} \in \mathfrak{P} : p_\infty = v_\infty\}} \xi^{\mathrm{wt}(\mathbf{p})} \omega_{\mathbf{p}}.$$

Proof. By Theorem 2.11, $\Psi^\ell | v_\infty \rangle$ is a column vector whose j^{th} component is an algebraic sum representing all feasible ℓ-paths $\mathbf{p}$ from v_j to v_∞ in the graph G in addition to all feasible ℓ-walks $\mathbf{p}$ from v_j to v_∞ which revisit the initial vertex v_j exactly once at some intermediate step. Computing the inner product with the vector $\langle \mathbf{1}_{\mathcal{A}_\mathfrak{C}}|$ thereby transforms the row vector into an algebraic sum of all components. Hence, $\langle \mathbf{1}_{\mathcal{A}_\mathfrak{C}} | \Psi^\ell | v_\infty \rangle$ is an algebraic sum representing all paths of length ℓ having terminal vertex v_∞. Summing over all possible path lengths then gives the stated result. □

The following notational convention for functionals and vectors associated with vertex subsets is adopted henceforth.

Definition 2.16. *For arbitrary vertex subset $W \subseteq V$, define the* $\mathcal{A}_\mathfrak{C}$-functional *$\langle W|$ by*

$$\langle W| := \sum_{w \in W} \langle \xi^{\mathbf{0}} \omega_{\{w\}}|.$$

Similarly, define the vector *$|W\rangle$ by*

$$|W\rangle := \sum_{w \in W} |w\rangle.$$

Given disjoint subsets of vertices $W_0, W_\infty \subset V$, the next corollary reveals a method of computing all feasible paths from initial nodes $v_0 \in W_0$ and terminal nodes $v_\infty \in W_\infty$. This allows one to compute feasible paths between *clusters* of nodes in a network.

Corollary 2.17. *Given a multi-weighted graph G on n vertices with nilpotent multi-weighted adjacency matrix Ψ, a constraint vector $\mathfrak{C} = (c_1, \ldots, c_m)$, a subset of vertices $W_0 \subseteq V$, and a subset of vertices $W_\infty \subseteq V$, the collection*

of all feasible paths with initial vertex $v_0 \in W_0$ and terminal vertex $v_\infty \in W_\infty$ in G is given by

$$\sum_{\ell=1}^{n} \langle W_0 | \Psi^\ell | W_\infty \rangle = \sum_{\{\mathbf{p} \in \mathfrak{C} : p_0 \in W_0, p_\infty \in W_\infty\}} \xi^{\mathrm{wt}(\mathbf{p})} \omega_{\mathbf{p}}.$$

Proof. The result follows from the proofs of Propositions 2.13 and 2.15. □

For purposes of implementation and clarity of pseudocode, the operator calculus (Dirac notation) formalism is simplified using basic linear algebra. For example, the linear functional $\langle x|$ is represented naturally by the *row vector* $\mathbf{x}$. The nilpotent adjacency operator Ψ associated with an n-vertex graph is naturally represented by an $n \times n$ matrix denoted unambiguously by Ψ.

Example 2.18. *Returning to the graph, the associated path-identifying nilpotent adjacency operator is represented by the following matrix:*

$$\begin{pmatrix} 0 & \overset{(.26,28)}{\xi}\ \omega_{\{2\}} & \overset{(.32,40)}{\xi}\ \omega_{\{3\}} & 0 & 0 \\ 0 & 0 & \overset{(.04,42)}{\xi}\ \omega_{\{3\}} & \overset{(.12,32)}{\xi}\ \omega_{\{4\}} & 0 \\ 0 & 0 & 0 & \overset{(.12,30)}{\xi}\ \omega_{\{4\}} & \overset{(.16,64)}{\xi}\ \omega_{\{5\}} \\ 0 & 0 & 0 & 0 & \overset{(.08,22)}{\xi}\ \omega_{\{5\}} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

where multi-exponents are written directly above the ξs in order to save space.

Recall that the task is to compute all feasible hop-minimal paths from v_1 to v_5, where each edge is weighted with a pair of additive weights subject to the constraints vector $(0.5, 100)$. Setting $\mathbf{v}_1 := \left(\overset{(0,0)}{\xi}\ \omega_{\{1\}}, 0, 0, 0, 0 \right)$, feasible 1-paths with initial vertex v_1 are recovered by computing the row vector

$$\mathbf{u} = \mathbf{v}_1 \Psi = \left(0, \overset{(.26,38)}{\xi}\ \omega_{\{1,2\}}, \overset{(.32,40)}{\xi}\ \omega_{\{1,3\}}, 0, 0 \right).$$

All feasible two-step paths with initial vertex v_1 are found by computing $\mathbf{u}\Psi = \mathbf{v}_1 \Psi^2$. The reader can verify that the fourth component of this vector is

$$(\mathbf{v}_1 \Psi^2)_4 = \overset{(0.38,60)}{\xi}\ \omega_{\{1,2,4\}} + \overset{(0.44,70)}{\xi}\ \omega_{\{1,3,4\}},$$

representing the two feasible 2-paths from v_1 to v_4. The reader can further verify that the fifth component of this vector is zero. The infeasible path $v_1v_3v_5$ has been "killed" by the properties of the algebra: $\overset{(.32,40)}{\xi}\overset{(.16,64)}{\xi} = 0$.

Finally, the one feasible three-step path from v_1 to v_5 is found by examining the 5th component of the row vector

$$\mathbf{v}_1\Psi^3 = \left(0, 0, 0, \overset{(0.42,100)}{\xi}\,\omega_{\{1,2,3,4\}}, \overset{(0.46,82)}{\xi}\,\omega_{\{1,2,4,5\}}\right).$$

The infeasible path $v_1v_3v_4v_5$ has been killed by the fact that in the constraints algebra, $\overset{(0.32,40)}{\xi}\overset{(0.12,30)}{\xi}\overset{(0.08,22)}{\xi} = \overset{(0.44,70)}{\xi}\overset{(0.08,22)}{\xi} = 0$. *The infeasible path $v_1v_2v_3v_5$ has been eliminated in similar fashion.*

3 Algorithms

Based on the results of the previous section, two types of algorithms are considered herein: centralized algorithms and distributed algorithms. In the centralized algorithms considered here, the graph's topology is known and fixed at all times. In the distributed case, the algorithm is implemented at a vertex, and only a subgraph or extended neighborhood is "visible" at any step of the algorithm.

Remark 3.1. *The algorithms appearing here are similar to, but more general than those put forth in Chapter 17 of the book by Mehdi and Ramasamy [12]. The OC approach accommodates more general constraints and simultaneously obtains all feasible hop-minimal (i.e., least-hop) paths for a given a source-destination pair.*

3.1 Centralized: Precomputed Routing

The first centralized algorithm considered here is applied to a fixed graph; i.e., the static case. Using the result of Theorem 2.11, Algorithm 1 enumerates all feasible hop-minimal paths from initial vertex v_0 to terminal vertex v_∞. Applying a choice function then allows the user to select a preferred path for the routing.

In the static case, the topology of the entire network is fixed and known. The static case centralized algorithm proceeds by developing all paths with source vertex v_0. At each iteration of the loop, a partial path is evolved one step. The algorithm terminates in two cases:

1. The partial paths all ultimately revisit vertices or violate the constraints. In this case the algorithm returns 0.

2. Any feasible path $v_0 \to v_\infty$ is found. At this point, the algorithm returns all existing feasible hop-minimal paths within the frame sequence considered.

Note that replacing the final line of Algorithm 1 by

$$\textbf{return } \daleth(\mathbf{u}_{v_\infty})$$

allows one to compute the *preferred* hop-minimal path from v_0 to v_∞.

Given initial vertex v_0, target vertex v_∞, and constrained path-identifying nilpotent adjacency operator Ψ, The centralized algorithm proceeds as follows.

1. Compute the action of the operator Ψ on the initial vertex v_0 and prepend the initial vertex to create a row vector, $\mathbf{u}$, containing partial paths of length 1. Simply stated, this is the v_0th row of Ψ with each entry left-multiplied by the element ω_{v_0}. In the pseudocode, this is denoted by $\mathbf{v}_0\Psi$.

2. If target has not been reached (v_∞-component of $\mathbf{u}$ is zero), apply the matrix Ψ to $\mathbf{u}$, thereby appending one step to each partial path.

3. Repeat step 2 until either the target has been reached or all partial paths have self-intersected, in which case $\mathbf{u} = 0$.

4. Return all existing hop-minimal paths to target (zero if no path exists). This information is found in the v_∞ component of $\mathbf{u}$.

Note that as a matrix, Ψ acts on row vectors by right-multiplication. Algorithm 1 expresses the centralized algorithm in pseudocode.

In the dynamic case, the matrix Ψ is replaced by a sequence of matrices $(\Psi_k : k \in \mathbb{N}_0)$. This sequence arises from discretization of the continuous-time evolution of the graph process. Since changes in graph topology occur at distinct points in time, the matrix Ψ_k represents a fixed graph topology valid from time t_{k-1} until time t_k. Distinct graphs of this sequence are referred to herein as *frames*. For convenience, it will be assumed that *consecutive matrices are distinct*; i.e, $\Psi_{t_k} \neq \Psi_{t_{k-1}}$ for $k \in \mathbb{N}$.

It will be assumed that the time is discretized and the graph sequence is adapted such that topology changes cannot occur during a hop. In particular, all partial paths can be advanced one step during any frame.

input : Source node v_0, target node v_∞, and constrained path-identifying nilpotent adjacency matrix Ψ.
output: Component of row vector $\mathbf{u}$ representing all multi-weighted paths of minimal length from v_0 to v_∞ satisfying the constraint vector.

$\mathbf{u} := \mathbf{v_0}\Psi$
while $[(\mathbf{u} \neq \mathbf{0}\,\mathbf{and}\,\mathbf{u}_{v_\infty} \neq 0)]$ **do**
  $\mathbf{u} := \mathbf{u}\Psi$
end
return $\mathbf{u}_{v_\infty}$

Algorithm 1: StaticCentralizedPaths (pseudocode)

The dynamic centralized algorithm proceeds by developing all paths with source vertex v_0. At each time step, a partial path is evolved one step or allowed to wait for the next time increment to advance. All partial paths are maintained and extended in subsequent frames. In addition to continuing partial paths, the algorithm allows the possibility that a new path from v_0 may be initiated in a new frame to become a feasible path $v_0 \to v_\infty$. The algorithm terminates when the time exceeds a final allowable time or when any feasible path $v_0 \to v_\infty$ is found. At that point, the algorithm returns all existing feasible hop-minimal paths within the frame sequence considered.

Given source node v_0, target node v_∞, initial time t_0, final allowable time t_∞, and a sequence of constrained path-identifying nilpotent adjacency operators $(\Psi_k : k \in \mathbb{N}_0)$, the dynamic algorithm proceeds as follows.

1. Initialize frame counter k and row vector $\mathbf{u} = \mathbf{v}_0\Psi_0$ of one-step partial paths originating at v_0.

2. Advance the frame counter and advance all partial paths one step while preserving all partial paths already obtained, by computing $\mathbf{u}(\mathbf{I} + \Psi_k)$. Here, $\mathbf{I}$ denotes the identity matrix.

3. Assuming network topology in frame k is different[1] from topology in frame $k-1$, also allow the possibility of new path starting in k^{th} frame by computing $\mathbf{u} := \mathbf{u} + \mathbf{v}_0\Psi_k$.

4. Repeat steps 2 and 3 until a path has been found to the target, v_∞, or until a maximum allowed number of frames have been considered.

[1]It is assumed that the process can be discretized in a manner to ensure this.

5. Return all feasible paths found or zero if no paths exist. This is accomplished by returning the v_∞ component of the row vector $\mathbf{u}$.

Once again, the matrices Ψ_k acts on row vectors by right-multiplication. Algebraically, steps 2 and 3 of the algorithm are conveniently handled by $\mathbf{u} := \mathbf{u}(\mathbf{I} + \Psi_k + \mathbf{v}_0\Psi_k)$. Algorithm 2 expresses the centralized dynamic algorithm in pseudocode.

input : Source node v_0, target node v_∞, initial time t_0, final allowable time t_∞, and a sequence of (not necessarily distinct) constrained path-identifying nilpotent adjacency matrices $(\Psi_k : k \in \mathbb{N}_0)$.
output: Algebraic element ψ representing multi-weighted hop-minimal paths from v_0 to v_∞ satisfying the constraint vector.

$k := 0$
$\mathbf{u} := \mathbf{v}_0\Psi_0$
while $[(t_k \leq t_\infty)$ **and** $(\mathbf{u}_{v_\infty} \neq 0)]$ **do**
  $k := k + 1$
  $\mathbf{u} := \mathbf{u}(\mathbf{I} + \Psi_k + \mathbf{v}_0\Psi_k)$
end
return $\mathbf{u}_{v_\infty}$

Algorithm 2: DynamicCentralizedPaths (pseudocode).

As before, replacing the final line of Algorithm 2 by

$$\textbf{return } \daleth(\mathbf{u}_{v_\infty})$$

allows one to compute the *preferred* hop-minimal path from v_0 to v_∞.

3.2 Distributed: Dynamic Routing

The distributed routing algorithm is implemented at individual nodes. Each node receiving a packet is able to choose a routing for passing the packet along, based on some partial information the node has about the current network topology. The distributed algorithm described here makes use of the following assumption:

While the topology of the graph is not known at any time, each node "knows" its own best-case minimal distance from every other node in the

underlying graph. This is the minimum number of hops between a node and every other node in the underlying graph assuming all links are valid. Writing $\mathcal{P}_{ij}$ for the set of all paths $\mathbf{p} : v_i \to v_j$ in the underlying graph G, let $d : V \times V \to \mathbb{N}_0 \cup \{\infty\}$ be the function defined by

$$d(v_i, v_j) := \begin{cases} \min\limits_{\mathbf{p} \in \mathcal{P}_{ij}} \{|\mathbf{p}|\} & \text{if } \mathcal{P}_{ij} \neq \emptyset, \\ \infty & \text{otherwise.} \end{cases}$$

In other words, $d(v_i, v_j)$ is the minimum number of hops required to reach v_j from v_i among all existing paths in the underlying graph. If no such path exists, $d(v_i, v_j) = \infty$.

In order to determine whether an evolving path is getting closer to the destination, a "distance oracle" function is employed. We define the *distance oracle* $\Delta : (\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n) \times (\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n) \to \mathbb{N}_0$ by

$$\Delta\left(\sum_{\mathbf{p} \in P_1 \subseteq \mathcal{P}} \xi^{\mathrm{wt}(\mathbf{p})} \omega_{\mathbf{p}}, \sum_{\mathbf{q} \in P_2 \subseteq \mathcal{P}} \xi^{\mathrm{wt}(\mathbf{q})} \omega_{\mathbf{q}}\right) := \min_{\substack{\mathbf{p} \in \mathcal{P}_1 \\ \mathbf{q} \in \mathcal{P}_2}} d(\mathbf{p}_{|\mathbf{p}|}, \mathbf{q}_{|\mathbf{q}|}).$$

Remark 3.2. *The distance oracle reveals the minimum number of hops required to reach the terminal vertex of a path in $\mathcal{P}_2$ from the terminal vertex of a path in $\mathcal{P}_1$.*

Given a finite path $\mathbf{b}$, denote by $\Psi_{(\mathbf{b}, t_k)}$ the $\mathbf{b}$-*neighborhood adjacency operator* at time t_k. This operator represents the adjacency of the *neighborhood* $\mathcal{N}(\mathbf{b}_{|\mathbf{b}|})$ in the graph at time t_k.

Algorithm 3 is implemented at each node of the underlying graph. It is worthwhile to note that computations are being performed in a sequence of algebras.

For convenience, define the canonical *path projection* $\pi_\Omega : \mathcal{A}_{\mathfrak{C}} \otimes \Omega_n \to \mathcal{P}$ by linear extension of

$$\pi_\Omega(\xi^{\mathrm{wt}(\mathbf{b})} \omega_{\mathbf{b}}) := \mathbf{b}.$$

Algorithm 4 simulates the node wise implementation to choose a preferred multi constrained path $v_0 \to v_\infty$ in the graph process $(G_{t_k} : k \in \mathbb{N}_0)$. Once again, it will be assumed that the time is discretized and the graph sequence is adapted such that topology changes cannot occur during a hop.

Algorithm 4 proceeds by developing a preferred path with source vertex v_0. At each time step, the partial path is augmented by one step such that the constraints are still satisfied and the absolute distance (number of hops)

input : Algebraic monomial $\xi^{\mathrm{wt}(\mathbf{b})}\omega_{\mathbf{b}} \in \mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$, target node v_∞, and discrete time t_k. The element $\xi^{\mathrm{wt}(\mathbf{b})}\omega_{\mathbf{b}}$ represents a multi-weighted partial path (of length $|\mathbf{b}|$) from fixed initial vertex v_0 to vertex $v_{\mathbf{b}_{|\mathbf{b}|}}$. The constraint vector is $\mathfrak{C} = (c_0, c_1, \ldots, c_m)$, in which the zeroth component, c_0, represents the maximum number of hops allowed to complete any path from $v_{\mathbf{b}_{|\mathbf{b}|}}$ to the target node.

Data: Neighborhood adjacency operator $\Psi_{(\mathbf{b},t_k)}$ valid at time step t_k is assumed to be available, as this is implemented at each node.

output: Algebraic element ψ representing a preferred multi-weighted path of length $|\mathbf{b}| + 1$ emanating from v_0, satisfying the fixed constraint vector, and whose last step does not increase the minimal distance to v_∞ over the previous partial path.

Get best-case remaining distance from target.
$\delta' := \Delta(\xi^{\mathbf{b}}\omega_{\mathbf{b}}, \xi^{\mathbf{0}}\omega_{v_\infty})$

Reset constraints so remaining distance cannot increase during path evolution.
$\mathfrak{C} := (\delta, c_1, \ldots, c_m)$

Extend partial path by one step, and return preferred feasible path.

return $\beth(\langle \xi^{\mathrm{wt}(\mathbf{b})}\omega_{\mathbf{b}} | \Psi_{(\mathbf{b},t_k)})$

Algorithm 3: ChoiceEvolve

from the target is not increased. The algorithm terminates when the time exceeds a final allowable time or when a feasible path $v_0 \to v_\infty$ is found.

Another useful distributed implementation allows multiple paths to evolve in parallel. By removing the choice function from Algorithm 3, the node-level implementation returns a collection of feasible partial paths whose terminal nodes are neighbors of the current node and whose maximal distance from the target does not exceed the current node's distance from the target. This algorithm, Algorithm 5 can then be applied recursively (by linear extension) to obtain a collection of feasible paths from a given source to a given target. An example of output obtained using Algorithm 5 can be seen in Figure 4.

input : Source vertex v_0, target node v_∞, initial time t_0, final allowable time t_∞, and neighborhood adjacency operator $\Psi_{(v_0,t_k)}$ valid at time step t_0.
OutputAlgebraic element ψ representing a feasible path $v_0 \to v_\infty$ if one is found, zero otherwise.

Initialize frame counter (k), best case distance (in hops) from source to target (δ), constraints, and preferred first step from v_0.
$k := 0$
$\delta := \Delta(\xi^{\mathbf{0}}\omega_{v_0}, \xi^{\mathbf{0}}\omega_{v_\infty})$
$\mathfrak{C} := (\delta, c_1, \ldots, c_m)$
$\psi := \beth\left(\langle \xi^{\mathbf{0}}\omega_{v_0} | \Psi_{(v_0,t_0)} | \mathbf{1}\rangle\right)$

Repeat until maximum frames considered or until remaining distance to target is zero.
while $\left[(t_k \le t_\infty)\ \textbf{and}\ [\Delta(\psi, \xi^0\omega_{v_\infty}) \neq 0]\right]$ **do**
  Advance frame counter.
  $k := k + 1$
  Extend partial path by one step.
  $\psi := \text{ChoiceEvolve}[\xi^{\mathbf{0}}\omega_{v_0}, v_\infty, t_k, \Psi_{(\pi_\Omega(\psi),t_k)}]$
end

Return preferred feasible path if one found. Otherwise, return zero.
return ψ

Algorithm 4: DistributedChoosePath

3.3 Remarks on complexity

Note that $\omega_i\langle v_i|\Psi^k$ is represented as a row vector whose nonzero entries represent all k-paths with initial vertex v_i. Similarly, $\Psi|v_j\rangle$ is represented by a column vector whose nonzero entries represent 1-paths with terminal vertex v_j. Computing the $(k+1)$-paths from v_i to v_j then requires computing $\omega_i\langle v_i|\Psi^k\Psi|v_j\rangle$. Letting $A = (a_{ij})$ be the ordinary adjacency matrix of G, recall that

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Letting λ denote the number of multiplications involved in this compu-

input : Algebraic monomial $\xi^{\mathrm{wt}(\mathbf{b})}\omega_{\mathbf{b}} \in \mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$, target node v_∞, and discrete time t_k. The element $\xi^{\mathrm{wt}(\mathbf{b})}\omega_{\mathbf{b}}$ represents a multi-weighted partial path (of length $|\mathbf{b}|$) from fixed initial vertex v_0 to vertex $v_{\mathbf{b}_{|\mathbf{b}|}}$. The constraint vector is $\mathfrak{C} = (c_0, c_1, \ldots, c_m)$, in which the zeroth component, c_0, represents the maximum number of hops allowed to complete any path from $v_{\mathbf{b}_{|\mathbf{b}|}}$ to the target node.

Data: Neighborhood adjacency operator $\Psi_{(\mathbf{b},t_k)}$ valid at time step t_k is assumed to be available, as this is implemented at each node.

output: Algebraic element ψ representing all feasible multi-weighted paths of length $|\mathbf{b}| + 1$ emanating from v_0, satisfying the fixed constraint vector, and whose last steps do not increase the minimal distance to v_∞ over the previous partial path.

Get best-case remaining distance from target.
$\delta' := \Delta(\xi^{\mathbf{b}}\omega_{\mathbf{b}}, \xi^{\mathbf{0}}\omega_{v_\infty})$

Reset constraints so remaining distance cannot increase during path evolution.
$\mathfrak{C} := (\delta, c_1, \ldots, c_m)$

Extend partial path by one step, and return feasible paths.
return $\langle \xi^{\mathrm{wt}(\mathbf{b})}\omega_{\mathbf{b}} | \Psi_{(\mathbf{b},t_k)}$

Algorithm 5: MultiplePathEvolve

tation,

$$\lambda = \sum_{\ell=1}^{n} a_{\ell j}(\sharp\{\text{feasible } k\text{-paths } v_i \to v_\ell\}) \\ \leq \sharp\{\text{feasible } k\text{-paths with source } v_i\}. \quad (3.5)$$

It follows immediately that the number of multiplications performed in determining $\omega_i\langle v_i|\Psi^k|v_j\rangle$ is bounded above by the number of feasible paths of length at most $k-1$ having initial vertex v_i. Hence, the next corollary is obtained.

Corollary 3.3. *Given a fixed pair of vertices v_0 and v_∞, the complexity of enumerating all feasible k-paths from v_0 to v_∞ with the constrained path-identifying nilpotent adjacency matrix is*

$$\mathcal{O}(n \cdot \sharp\{\mathbf{p} \in \mathcal{F}_{v_0} : |\mathbf{p}| \leq k-1\}),$$

where $\mathcal{F}_{v_0}$ denotes the collection of all feasible paths having initial vertex v_0.

The computational complexity stated above is in terms of monomial multiplications performed within the algebra. Recall that for disjoint ordered multi-indices $\mathbf{p}$, $\mathbf{q}$, the product $\omega_{\mathbf{p}}\omega_{\mathbf{q}} = \omega_{\mathbf{p}.\mathbf{q}}$ is given by sequence concatenation. Moreover, there are m component-wise binary operations performed on the multi-exponents representing weights. Hence, some additional polynomial cost is associated with the implementation of this algebra multiplication.

4 Application: Multi-constrained routing in a wireless mesh sensor network

The problem posed by Ben Slimane, *et al.* involves computing feasible paths in wireless mesh sensor networks (WMSNs) subject to three additive constraints as described in [2]. Wireless mesh sensor networks (WMSNs) may consist of a collection of spatially distributed sensor nodes, characterized by limited memory, processing capability and battery power supply which are organized in a full mesh topology. WMSNs are expected to support various types of applications with different QoS requirements. According to novel application requirements, QoS constraints become more and more critical in terms of end-to-end delay, data throughput and packet-error- rate. Also, due to energetic constraints at node level, energy saving remains the most challenging issue.

The authors of [2] propose a cross-layer algorithm for WMSNs, referred to as the Joint duty cycle Scheduling, resource Allocation and multi-constrained QoS Routing algorithm (JSAR). To deal with different WMSNs requirements, JSAR combines simultaneously, a duty cycle scheduling scheme for energy saving, a resource (time slots per available frequency channels) allocation scheme for resource efficiency, and a heuristic multi-constrained routing protocol for multi-constrained QoS support. They go on to implement JSAR and evaluate its performance, showing that JSAR performs better with the increase of the number of available frequency channels and time slots.

An example on a 32-node graph found in that work is considered herein using OC methods. The OC approach is applied in the static case to enumerate (precompute) all hop-minimal feasible paths from a given source to a given target in the graph. The approach assumes the topology of the entire graph is available and fixed throughout the process. Sample paths obtained and summaries of running times are presented below.

The OC approach is applied in the distributed case to enumerate hop-minimal feasible paths from a given source to a given target in the graph subject to the requirement that each step of the path not increase the distance (in number of hops) from the destination. The approach assumes that each vertex knows (and reports) its minimal hop distance from the destination, and that each vertex can "see" (i.e. query) all of its neighbors.

The distributed algorithm is applied to the 32-node static case first and a dynamic version of the 32-node example second. Sample paths obtained and summaries of running times appear in Section 4.1.

Figure 2 depicts a weighted graph on 32 vertices, as considered in [2]. Each edge is assigned a vector of positive weights in $\mathbb{R}^3$. Simulations were processed using five different examples of the constraint vector $\mathfrak{C} = (c_1, c_2, c_3)$ appearing in Table 1, where:

- The first constraint c_1 (Max Packet Reception Rate) represents the upper bound of the logarithmic value of the packet reception rate of a feasible path. The link's logarithmic value of the packet reception rate prr_{uv} is defined as the absolute value of the logarithmic value of PRR_{uv} (packet reception rate).

$$PRR_{s\to d} = \sum_{l_{uv}\in P_{s\to d}} prr_{uv} = \sum_{l_{uv}\in P_{s\to d}} |log_{10}(PRR_{uv})|$$

- The second constraint c_2 (Max Delay) represents the upper bound of the end-to-end delay of a feasible path.

$$\delta_{s\to d} = \sum_{l_{uv}\in P_{s\to d}} \delta_{uv}$$

- The third constraint c_3 (Max Energy) represents the upper bound of the total consumed energy of a feasible path. E^c_{uv} represents the sum of the energy consumed by node u during radio transmission and data processing and queueing (E^c_u) and the energy consumed by node v during radio receiving (E^c_v).

$$E^c_{s\to d} = \sum_{l_{uv}\in P_{s\to d}} E^c_{uv}$$

The constraint vector C_i represents stricter constraints than the constraint vector C_{i+1}.

Constraints	Max Packet Reception Rate	Max Delay	Max Energy
$\mathfrak{C}_1$	1.61604	2.41987	68.1328
$\mathfrak{C}_2$	2.83083	4.75579	185.967
$\mathfrak{C}_3$	3.91719	6.85752	330.864
$\mathfrak{C}_4$	4.75373	10.1413	428.139
$\mathfrak{C}_5$	5.07585	14.1619	643.696

Table 1: Constraints considered by Ben Slimane.

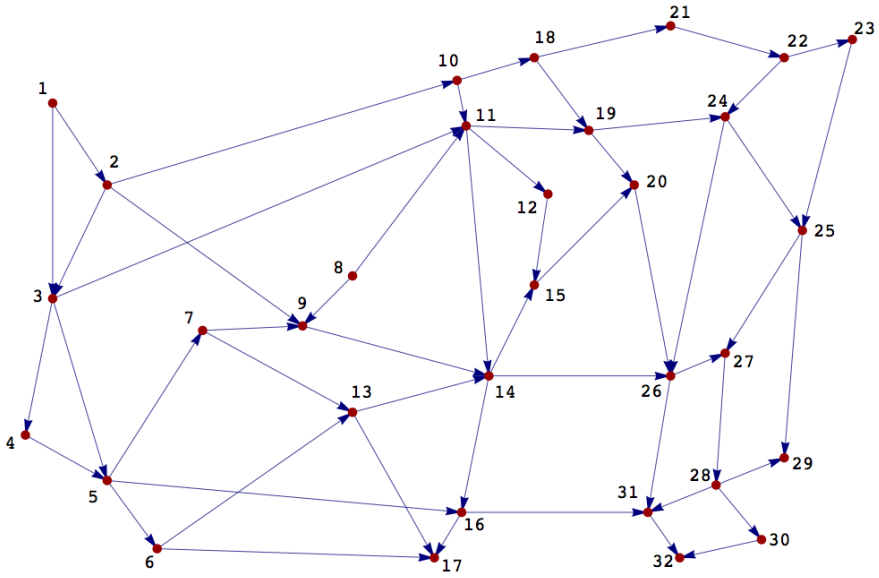

Figure 2: Graph on 32 vertices originally considered by Ben Slimane, *et al.*

Note that for fixed $\mathfrak{C} \in \mathbb{R}^3$ with additive weights, multi-exponents $\mathbf{x} = (x_1, x_2, x_3)$ appearing among basis elements $\xi^{\mathbf{x}} \in \mathcal{A}_{\mathfrak{C}}$ must satisfy $\mathbf{x} \preceq \mathfrak{C}$. Given $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$, multiplication of arbitrary algebraic monomials consequently satisfies

$$\xi^{\mathbf{x}}\xi^{\mathbf{y}} = \begin{cases} \xi^{\mathbf{x}+\mathbf{y}} & \text{if } \mathbf{x}+\mathbf{y} \preceq \mathfrak{C}, \\ 0 & \text{otherwise.} \end{cases}$$

In order to apply the OC approach to problems of identifying paths satisfying multiple constraints (represented by $\mathfrak{C}$), the path-identifying nilpotent adjacency matrix will be extended by allowing entries from the algebra $\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$. In this approach, a path $\mathbf{u} = (u_1, \ldots, u_m)$ of (additive) weight $\mathrm{wt}(\mathbf{u}) = \mathbf{x} \in \mathbb{R}^3$ will be represented in $\mathcal{A}_{\mathfrak{C}} \otimes \Omega_n$ by an element of the form

$\xi^{\mathrm{wt}(\mathbf{u})}\,\omega_{\mathbf{u}}$. The concatenation of this path with another path $\mathbf{v} = (v_1, \ldots, v_\ell)$ of weight $\mathrm{wt}(\mathbf{v}) = \mathbf{y} \in \mathbb{R}^3$ is then represented by the product

$$(\xi^{\mathrm{wt}(\mathbf{u})}\omega_{\mathbf{u}})(\xi^{\mathrm{wt}(\mathbf{v})}\omega_{\mathbf{v}}) = \begin{cases} \xi^{\mathrm{wt}(\mathbf{u})+\mathrm{wt}(\mathbf{v})}\omega_{\mathbf{u}\,.\mathbf{v}} & \text{if } \mathbf{u}\,.\mathbf{v} \in \mathfrak{P}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that for the centralized process, the multiplicative identity in the constraints algebra is $\xi^{\mathbf{0}} := \xi^{(0,0,0)}$.

Formally, a *request* is a triple $(v_0, v_\infty, \mathfrak{C})$ consisting of a source, a target, and a constraints vector. Informally, such a triple represents a request to send a packet from a source to a destination subject to constraints determined by $\mathfrak{C}$. The routing algorithm processes a request in order to find an optimal path satisfying all of the constraints.

The algorithm works by finding all feasible paths emanating from a single source. The algorithm stops when any feasible path from the source to the destination is found. At that point, the algorithm returns *all* feasible paths from the source to the destination requiring the same number of hops. These paths are referred to as being *hop-minimal* feasible paths. No shorter feasible path exists.

Alternatively, the algorithm stops when no feasible paths exist (all possibilities are explored or excluded). In this case, the algorithm returns zero, and one knows that *no feasible path exists*.

Examples 4.1 and 4.2 were computed using *Mathematica* 8 for MAC OS X on a MacBook Pro with 2.4 GHz Intel Core i7 processor and 8 GB of 1333 MHz DDR3 SDRAM.

Example 4.1. *Figure 3 depicts running times of precomputing (centralized) feasible paths in the graph of Figure 2 over 100 randomly-generated requests. The trials were repeated five times subject to each of the five constraint vectors proposed in Table 1. Smaller points plotted correspond to tighter constraints. As expected, tighter constraints lead to shorter running times since fewer paths are developed.*

4.1 Distributed implementation

In the distributed example, the algorithm is implemented at a node. The node is aware of its adjacent neighbors, and each node knows its own minimal hop distance from a fixed destination. Note that this minimum distance is the minimum path length obtained by applying either Dijkstra's algorithm or the Bellman Ford algorithm to a graph whose edges are all of unit weight.

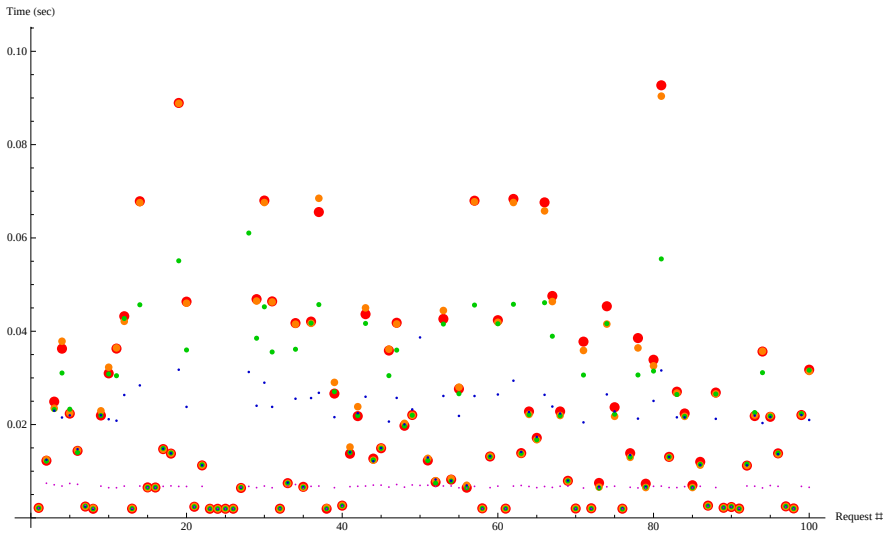

Figure 3: 100 trials of (centralized) precomputed routing over 5 constraint vectors of Table 1. Smaller points represent tighter constraints.

The node-level implementation is done using a single row of a path-identifying nilpotent adjacency matrix, because the full structure of the graph is assumed to be unknown. In the simulations run here, however, all such row vectors are stored as a single matrix. Two types of simulations are run here: a single path distributed process and a multiple path parallel distributed process.

In the single path process, the node-level implementation results in a single choice for routing of a packet in the next step. The choice is made such that remaining distance (in hops) to the target is minimal and does not increase over the current step. This is the direct implementation of Algorithm 4. The algorithm runs quickly, but sometimes fails to find existing feasible paths, since it only returns paths whose sequences of remaining distances are monotonically decreasing.

In the multiple path parallel process, the node-level implementation returns feasible partial paths to all of the current nodes neighbors whose distances to the target are not greater than the current nodes distance to the target. In this way, feasible multiple paths can be obtained while still exhibiting efficient runtimes.

In the distributed process, the multiplicative identity in $\mathcal{A}_{\mathfrak{C}}$ is $\xi^{(\infty,\mathbf{0})} := \xi^{(\infty,0,0,0)}$. When applied to the static case, this implementation offers even better runtimes than the previous experiments by pruning inefficient paths

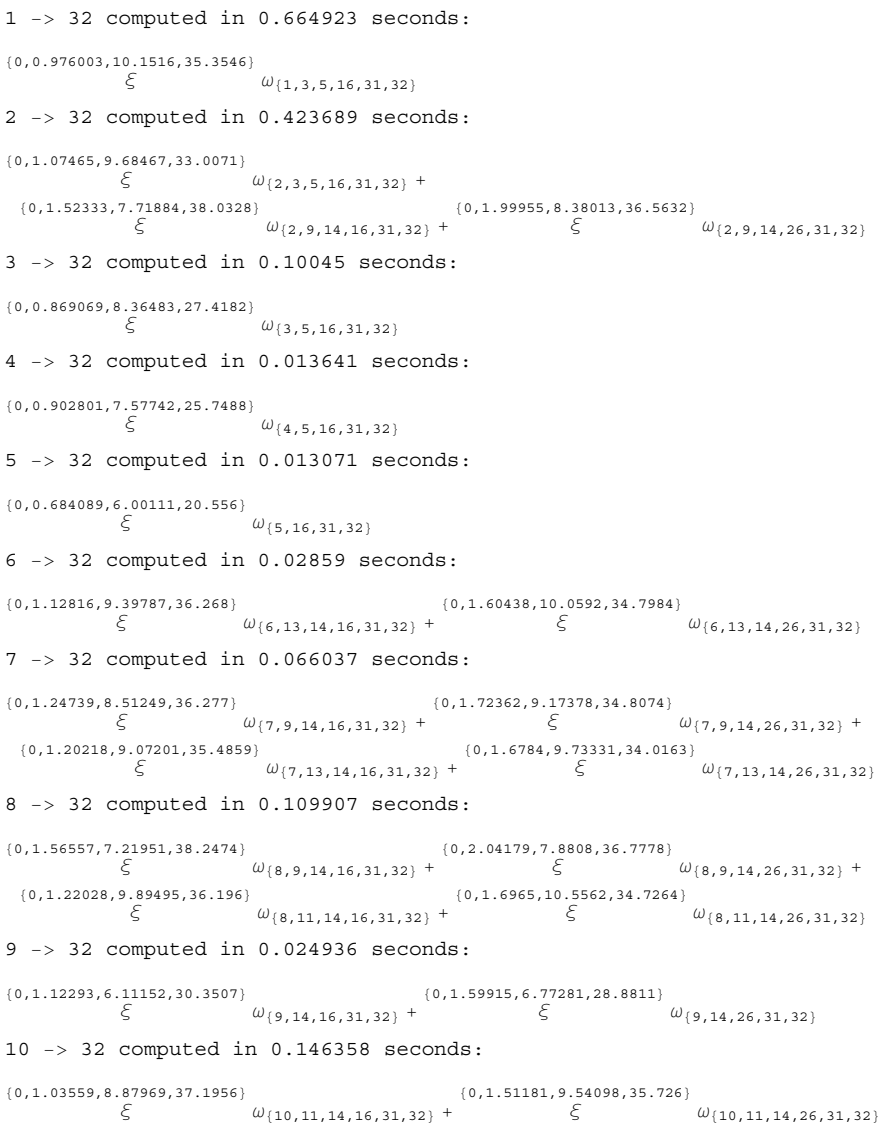

Figure 4: Multiple path parallel distributed routing. Enumerate feasible paths to v_{32} in graph of Figure 2. Paths satisfy constraint vector C5 of Table 1.

and eliminating the full matrix multiplication.

The distributed implementation has the advantage of making the dynamic case much easier to model than in the full adjacency matrix implementation. Some code has already been developed to consider intermittent links and

nodes. In Figure 4, feasible paths $v_0 \rightarrow v_{32}$ are enumerated in parallel over sources $v_0 \in \{v_1, \ldots, v_{16}\}$ in the graph of Figure 2.

Example 4.2. *Sample output can be easily obtained for larger graphs. For purposes of comparison, the 128-node graph of Figure 5 was generated in a manner to guarantee a degree of similarity with the 32-node example provided by Ben Slimane,* et al. *In particular, the graph was generated such that each node has out-degree between zero and seven to keep the average out-degree relatively low.*

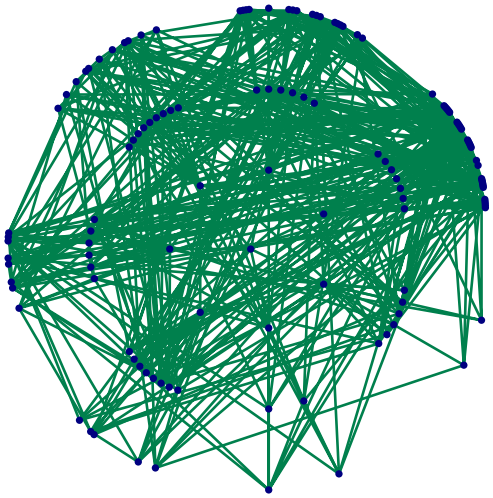

Figure 5: Randomly generated 128-node graph.

In Figure 6, runtimes of multiple-path distributed routing on 128-node graph of Fig. 5 are depicted. For each constraint vector of Table 1, 100 randomly-generated requests were processed. Smaller points correspond to tighter constraints.

4.2 Comparison between OC and SAMCRA

In this section, we present a performance comparison between the centralized version of OC and SAMCRA, implemented in Java. Both algorithms pre-compute the paths and solve the multi-constrained optimization problem.

SAMCRA returns one path between a nodes pair. This path has the minimum non-linear length ℓ that satisfies all the constraints. If we consider a path P with k links: $P = \{e_1, e_2, \ldots, e_k\}$, the length of P is [10]:

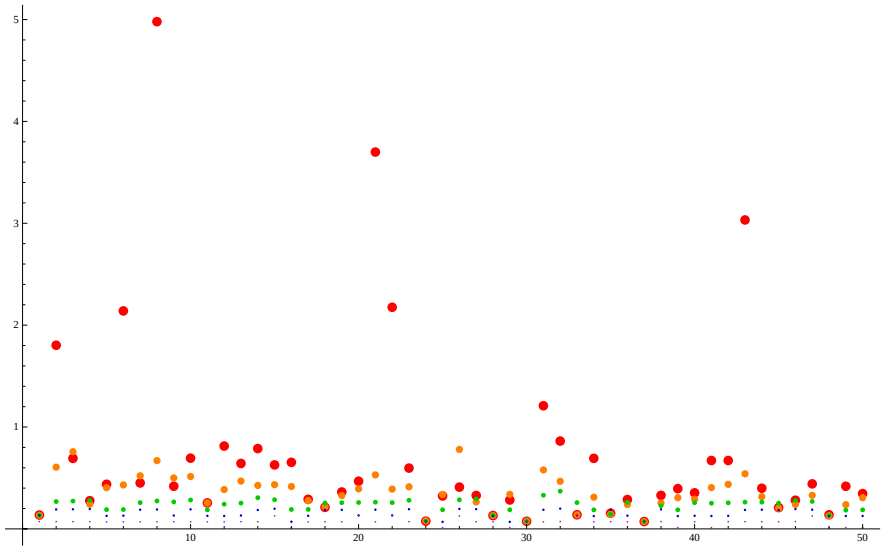

Figure 6: Runtimes of multiple-path distributed routing on 128-node graph of Fig. 5 using 100 randomly-generated requests over five constraint vectors of Table 1. Smaller points represent tighter constraints.

$$\ell(P) = \max_{1 \le i \le m} \left(\frac{\sum_{j=1}^{k} W_i^{e_j}}{L_i} \right)$$

where m is the number of weights and L the constraint vector.

SAMCRA is based on two more principles: the $k-$shortest path approach and the non-dominance. The k-shortest path approach is used when more than one possible paths between a source and a destination is needed to be found. In SAMCRA, this approach is applied only to the intermediate nodes, where we store not only one but multiple sub-paths from the source to each intermediate node, only when these paths are non-dominated (I.1) (I.2). In each step, SAMCRA chooses the sub-path with the minimum path length and stops whenever the first feasible path is found.

Using the "nilpotent adjacency matrix", OC can find a set of feasible paths between any source and destination or between a nodes pair. However, it can stop whenever any feasible path is found and return all the hop-minimal feasible paths. In each step, the multiplication of the matrices evolves the sub-paths one hop, if these paths satisfy the constraints.

OC and SAMCRA are exact, so they are able to find a solution, if a solution exists. SAMCRA is relied on Dijkstra, so it is limited on working with positive edge weights, while OC provides flexibility working with both

positive and negative weights. Moreover, SAMCRA can deal with min(max) QoS metrics by checking all the links of the graph in advance and omitting those that do not satisfy the constraints. OC deals with min(max) metrics in the main process.

Tables II and III, depict the total computation time of 100 random requests of SAMCRA and OC on the 32-node graph of Fig. IV.I and 128-node graph of Fig. IV.5., using the constraint vector C_3 of Table 1. OC stops when any feasible path from the source to the destination is found. In SAMCRA, Dijkstra is implemented from all the nodes to all the other nodes. In [10], it is proposed to apply Dijkstra from the destination to the rest of the nodes for efficiency but this is not possible when the graph is directed.

The tables reveal that SAMCRA requires more execution time than OC for computing 100 random requests. The complexity of both algorithms depends on the number of nodes and therefore as larger is the graph as more time the algorithms need to be executed. The tables also reveal that in the small graph of 32 nodes the difference of runtimes between the two algorithms is not remarkable but in the bigger graph of 128-nodes, OC is much more efficient.

Fig. IV.7 and IV.8 depict precisely the runtimes of each request, using the constraint vector C_3. A polynomial curve fitting is used ('poly2' in Matlab). OC outperforms SAMCRA in all cases.

	C_1	C_2	C_3	C_4	C_5
SAMCRA	41.6	35.8	30.6	24.1	24.5
OC	24.1	21.3	20.1	21.7	19.7

Table 2: Execution time in msec of 100 (centralized) random requests over the graph of Fig. IV.1

	C_1	C_2	C_3	C_4	C_5
SAMCRA	5262.9	5262.4	5398.3	5449.5	5438.0
OC	278.9	177.6	51.6	44.9	44.6

Table 3: Execution time in msec of 100 (centralized) random requests over the graph of Fig. IV.5

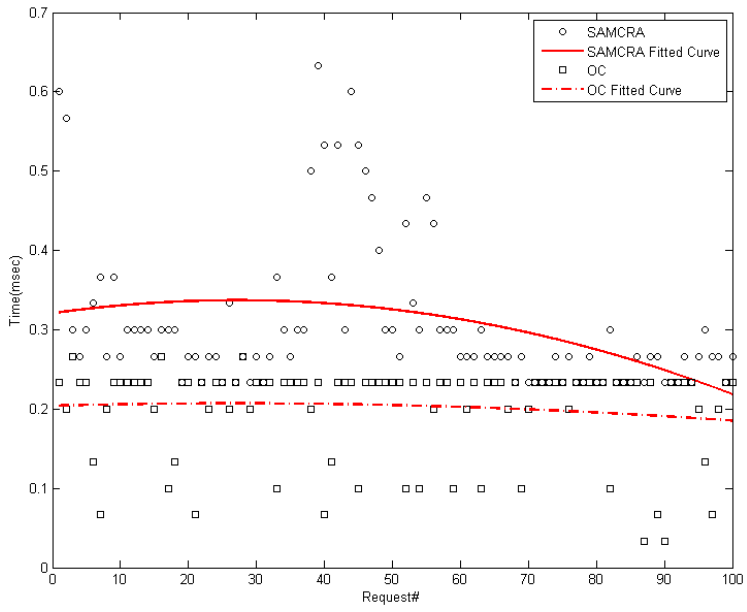

Figure 7: Runtimes of OC and SAMCRA on 32-node graph of Fig. IV.1 over the constrained vector C_3.

4.3 Comparison between OC and Sobrino's algorithm (D-OTF)

According to Sobrino, an optimal path is the lexicographic-lightest path obtained by strict isotonicity. Classical algorithms that are trying to solve the shortest path problem, return an optimal path in terms of an additive weight (delay, hop count etc) which is the minimum path for a source-destination pair. D-OTF ensures that any subpath of an optimal path has to be optimal as well.

Solving the shortest path problem, Sobrino's algorithm does not consider multiple metrics neither accept lower and upper bound cost requirements during the graph search. It does not work with negative weights as well. Trying to compare OC and Sobrino's algorithm, in case of the latter we reduce the problem using one positive metric in each link (the delay) without considering any constraint. D-OTF returns one path and OC stops whenever one path is found using the constraint vector C_3. Fig. IV.9 and IV.10 reveal that in that case Sobrino's algorithm is the best, demanding the least

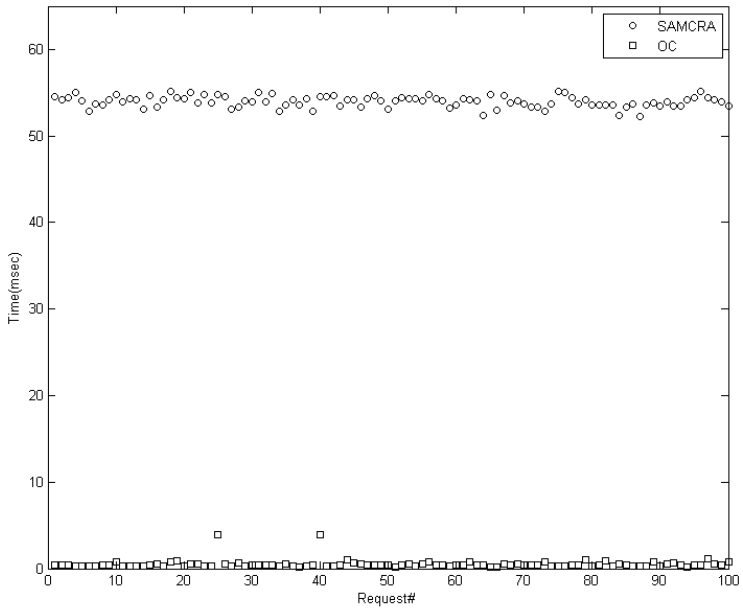

Figure 8: Runtimes of OC and SAMCRA on 128-node graph of Fig. IV.5 over the constrained vector C_3.

execution time since it has the same complexity as Dijkstra. Nevertheless, it can not be applied for solving the multi-constrained path problem.

All three algorithms (centralized OC, SAMCRA and D-OTF) have been implemented in Java(VM) 1.6.0.27 and computed in Ubuntu 12.04.3 with 2x3 GHz Intel Core 2 Duo processor and 8 GB DDR2.

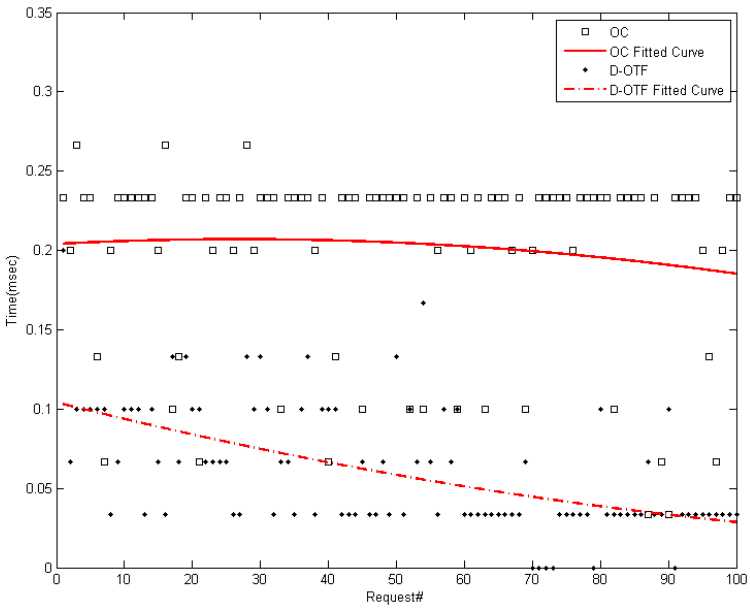

Figure 9: Runtimes of OC and D-OTF on 32-node graph of Fig. IV.1 over the constrained vector C_3.

5 Conclusion

The algorithms presented here are "quantum-like" in the sense that they are expressed using sequences of operators defined on algebras. Unlike the case in quantum computing, these operators are implemented using programming languages like *Mathematica* and *Java*, eliminating any need for quantum error-correction. The strengths of the OC approach lie in its flexibility and simplicity.

A number of avenues for further study exist, including the following:

1. Consider additional constraints in the static case. Notably, we want to consider probability of link existence, so that each path will have a probability of viability appearing in its overall weight. This will make path selection easier.

2. Compare with explicit paths generated by other methods.

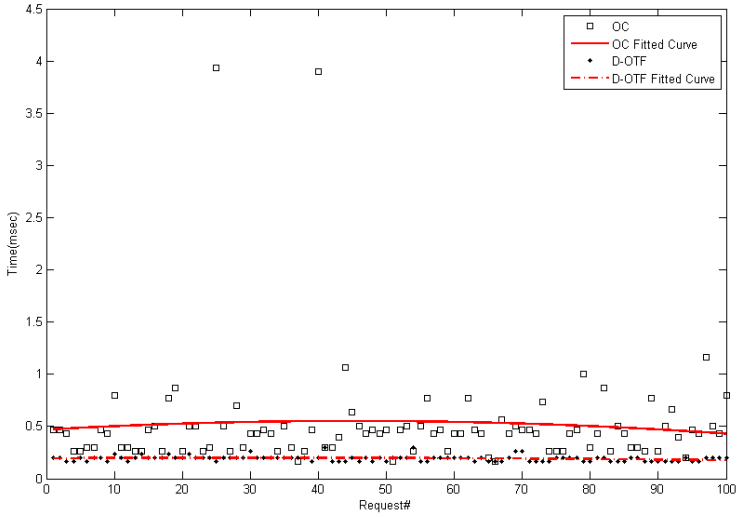

Figure 10: Runtimes of OC and D-OTF on 128-node graph of Fig. IV.5 over the constrained vector C_3.

3. Solving the two disjoint path problem using OC approach.

4. Further develop the distributed and dynamic distributed implementations for this problem. The OC approach offers a great deal of flexibility. For example, considering multi-hop visibility in the distributed algorithm is of particular interest.

References

[1] R. Bellman, On a routing problem, Quarterly of Applied Mathematics, **16**, (1958),87-90.

[2] J. Ben Slimane, Y-Q. Song, A. Koubaa, M. Frikha, Joint duty cycle Scheduling, resource Allocation and multi-constrained QoS Routing algorithm, Lecture Notes in Computer Science, **6811**, Ad-hoc, Mobile, and Wireless Networks, pages 29-43, 10th International Conference, ADHOC-NOW 2011, Paderborn, Germany, 2011.

[3] H. W. Corley, I.D. Moon, Shortest paths in networks with vector weights, Journal of Optimization Theory and Applications, **46,** (1985), 79-86. http://dx.doi.org/10.1007/BF00938761

[4] H. Cruz-Sánchez, G. S. Staples, R. Schott, Y-Q. Song, Operator calculus approach to minimal paths: Precomputed routing in a store-and-forward satellite constellation, Proceedings of IEEE Globecom 2012, Anaheim, USA, December 3-7, 2012, 3438–3443.

[5] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik, **1,** (1959), 269-271. http://dx.doi.org/10.1007/BF01386390

[6] L. R. Ford Jr., D. R. Fulkerson, Flows in Networks, Princeton University Press, 1962.

[7] L. Fu, D. Sun, L. R. Rillett, Heuristic shortest path algorithms for transportation applications: State of the art, Computers & Operations Research, **33,** (2006), 3324–3343.

[8] R. Hou, K.-S. Lui, K.-C. Leung, F. Baker, An Approximation Algorithm for QoS Routing with Two Additive Constraints, Proc. IEEE ICNP 2008, 328–337.

[9] H. A. Karimi, Real-time optimal route computation: a heuristic approach, ITS Journal, **3,** (1996), 111–127.

[10] F. A. Kuipers, Quality of Service Routing in the Internet: Theory, Complexity and Algorithms, Delft University Press, The Netherlands, 2004.

[11] G. Liu, R. Ramakrishnan, A*Prune: an algorithm for finding K shortest paths subject to multiple constraints, Proceedings of INFOCOM 2001, Anchorage, Alaska, April 2001, 743-749.

[12] D. Medhi, K. Ramasamy, Network Routing: Algorithms, Protocols, and Architectures, Morgan Kaufmann/Elsevier, 2007.

[13] R. Schott, G.S. Staples, Operator Calculus on Graphs (Theory and Applications in Computer Science), Imperial College Press, London, 2012.

[14] R. Schott, G.S. Staples, Operator calculus on generalized zeon algebras: theory and application to multi-constrained path problems, Prépublications de l'Institut Elie Cartan,

2011/18, (2011). Online at http://hal.archives-ouvertes.fr/hal-00603748/PDF/staceyQoSjune11.pdf.

[15] R. Schott, G. S. Staples, Nilpotent adjacency matrices and random graphs, Ars Combinatoria, **98,** (2011), 225-239.

[16] R. Schott, G.S. Staples, Nilpotent adjacency matrices, random graphs, and quantum random variables, J. Phys. A: Math. Theor., **41,** (2008), 155205.

[17] J. L. Sobrinho, Algebra and algorithm for QoS path computation and hop-by-hop routing in the Internet, IEEE ACM Transactions on Networking, **10,** (2002), 541-550. http://dx.doi.org/10.1109/TNET.2002.801397

[18] G. S. Staples, Clifford-algebraic random walks on the hypercube, Advances in Applied Clifford Algebras, **15,** (2005), 213-232.

[19] G. S. Staples, A new adjacency matrix for finite graphs, Advances in Applied Clifford Algebras, **18,** (2008), 997-991.

[20] P. Van Mieghem, F. A. Kuipers, Concepts of Exact QoS Routing Algorithms, IEEE/ACM Transactions on Networking, **12,** (2004), 851–864.

[21] P. Van Mieghem, F. A. Kuipers, On the complexity of QoS routing, Computer Communications, **26,** (2003), 376–387.

[22] Z. Wang, J. Crowcroft, QoS routing for supporting resource reservation, IEEE Journal on Selected Areas in Communications, **14,** (1996), 1228–1234.

[23] G. Xue, W. Zhang, J. Tang, K. Thulasiraman, Polynomial time approximation algorithms for multi-constrained QoS routing, IEEE/ACM Transactions on Networking, **16,** (2008), 656–669.