Efficient Exact Paths For Dyck and semi-Dyck Labeled Path Reachability*

Phillip G. Bradford†

February 15, 2018

Abstract

The *exact path length problem* is to determine if there is a path of a given fixed cost between two vertices. This paper focuses on the exact path problem for costs $-1, 0$ or $+1$ between all pairs of vertices in an edge-weighted digraph. The edge weights are from $\{-1, +1\}$. In this case, this paper gives an $\widetilde{O}(n^{\omega})$ exact path solution. Here ω is the best exponent for matrix multiplication and $\widetilde{O}$ is the asymptotic upper-bound mod polylog factors.

Variations of this algorithm determine which pairs of digraph nodes have Dyck or semi-Dyck labeled paths between them, assuming two parenthesis. Therefore, determining digraph reachability for Dyck or semi-Dyck labeled paths costs $\widetilde{O}(n^{\omega})$. A path label is made by concatenating all symbols along the path's edges.

The exact path length problem has many applications. These applications include the labeled path problems given here, which in turn, also have numerous applications.

1 Introduction

Shortest path algorithms are a great success. Many people use them and many vehicles are equipped with them. Determining path reachability is also important. Path reachability is often computed using transitive closure.

This paper efficiently solves the 0 and ±1 exact path length problem for digraphs whose edges have weights from $\{-1, +1\}$.

*This current paper uses standard definitions of Dyck and semi-Dyck languages. The author's earlier abstracts reversed the Dyck and semi-Dyck definitions. An extended abstract of this paper is in the proceedings of the UEMCON 2017 conference, see [1].

†Department of Computer Science and Engineering, University of Connecticut, Stamford, CT, USA. phillip.bradford@uconn.edu, phillip.g.bradford@gmail.com

Context-free language constrained graph problems are fundamental to a plethora of challenges. This paper gives algorithms for determining Dyck (semi-Dyck) constrained paths on digraphs based on the exact path problem. Dyck and semi-Dyck context-free languages are important. A central application for the exact path problem is for determining Dyck and semi-Dyck constrained paths in digraphs. Here these languages have a single parenthesis type.

Definition 1 (Exact path length problem [2]) Consider an integer edge weighted digraph G. Given an integer κ, the **EPL** *(exact path length problem)* is to determine whether there is a path between a given pair vertices costing exactly κ.

Nykänen and Ukkonen [2] show the general EPL is $\mathcal{NP}$-Complete. They also give a pseudo-polynomial algorithm for the EPL. The current paper uses a special case of the EPL where $\kappa \in \{-1, 0, +1\}$ and edge costs are from the set $\{-1, +1\}$.

Given these restricted edge costs, and for $\kappa \neq 0$, applying Nykänen and Ukkonen's algorithm costs $O(n^3 + n^\omega \log |\kappa|)$ time, see [2][1]. For $\kappa = 0$, their algorithm costs $O(n^3)$.

Solving this Dyck (semi-Dyck) labeled path problem is interesting due to the close relationship between transitive closure, Boolean and algebraic matrix multiplication, and context-free grammar recognition. For example, Lee [3] gives an equivalence between Context-free parsers and Boolean matrix multiplication algorithms.

1.1 Semi-Dyck and Dyck Constrained Graphs

Dyck and semi-Dyck languages are parenthesis languages. Dyck or semi-Dyck languages with two parenthesis symbols and n total parentheses can be parsed in $O(n)$ time and space. However, efficiently computing Dyck (and semi-Dyck) constrained reachability on digraphs seems more challenging.

Let $\mathcal{D}$ be a Dyck language of one open-parentheses symbol $\mathbf{a}$ and one close-parentheses symbol $\mathbf{a^{-1}}$. A sentence $w \in \mathcal{D}$ iff w can be reduced using right-inverse reduction, e.g. $\mathbf{a}\,\mathbf{a^{-1}} = \epsilon$, to the empty string ϵ. The Dyck language $\mathcal{D}$ is derivable from the grammar:

$$\mathcal{D} \implies \epsilon \mid \mathcal{D}\,\mathcal{D} \mid \mathbf{a}\,\mathcal{D}\,\mathbf{a^{-1}}.$$

Semi-Dyck languages allow reductions using both right-inverses and left-inverses $\mathbf{aa^{-1}} = \mathbf{a^{-1}a} = \epsilon$. They are derivable from the grammar:

$$\mathcal{S} \implies \epsilon \mid \mathcal{S}\mathcal{S} \mid \mathbf{a}\mathcal{S}\mathbf{a^{-1}} \mid \mathbf{a^{-1}}\mathcal{S}\mathbf{a}.$$

Dyck languages generate all strings of balanced parenthesizations. Semi-Dyck languages generate all strings of equal numbers of matching symbols.

The next definition is similar to one in [4].

[1]All logs are base 2 except where specified otherwise.

Definition 2 (Labeled Directed Graph) A *labeled directed graph* (LDG) is a multigraph $G = (\Sigma, V, E_1)$ consisting of a set V of vertices and a set $E_1 \subseteq V \times V \times \Sigma$ of labeled and directed edges.

The set Σ contains a grammar's terminals. If the grammar is Dyck (semi-Dyck), then Σ is said to be Dyck (semi-Dyck).

Given Definition 2, restrict cycles to having no repeated edges. LDGs are multigraphs. All LDG edges are augmented with *label-costs*. So each edge e in G has a label $l(e)$ and a label-cost $\boldsymbol{lc}(e)$. The label-cost function is,

$$\boldsymbol{lc}(e) \quad = \quad \begin{cases} -1 & \text{if } l(e) = a^{-1} \\ +1 & \text{if } l(e) = a. \end{cases}$$

A $+1$ edge and a -1 edge may be joined to form a new 0 label-cost edge for computing an exact path. After some processing, say such a new 0 label-cost edge e is created. Then, the label-cost function extends so $\boldsymbol{lc}(e) = 0$. This new 0 label-cost edge is added to an augmented edge set in the LDG. Also, ± 1 and 0 label-cost edges may be extended by adjoining 0 label-cost edges.

The label-costs or costs are written above edges such as $e = i \xrightarrow{w} j$. Therefore, in general $w \in \{-1, 0, +1\}$, but at the start of our algorithms assume $w \in \{-1, +1\}$.

1.2 Previous Work

Greenlaw, Hoover, and Ruzzo [5] discuss several formal-language based reachability problems. See also Afrati and Papadimitriou [6], Reps [7], and Ullman and Van Gelder [8]. For example, the *LGAP (labeled graph accessibility problem)* [5] is a Dyck language with constrained reachability problem on a directed graph G that is $\mathcal{P}$-complete when $|\Sigma| \geq 4$. Yannakakis [9, p. 237] points out that Valiant's Boolean matrix multiplication context-free word recognition algorithm determines single-source labeled path reachability in DAGs. This means there is an algorithm costing $O(n^\omega \log n)$ for finding context-free labeled and unweighted paths in DAGs with n vertices, where ω is the best exponent for $n \times n$ matrix multiplication. Very efficient matrix multiplication algorithms include results of Coppersmith and Winograd [10]; Stothers [11]; Williams [12]; and Le Gall [13]. Currently, the best exponent of square matrix multiplication is $\omega < 2.373$.

Melski and Reps [14] give an $O(|S|^3 n^3)$ context-free language reachability algorithm. Where S is the set of terminals and non-terminals for the input grammar. Barrett, Jacob, and Marathe [4] give an $O(n^3 |R||N|)$ algorithm for finding the all-pairs shortest paths in context free grammar constrained path problems. Here R is the set of rules and N is the set of non-terminals in Chomsky normal form.. This algorithm does not compute shortest paths with negative edge weights.

Alon, Galil, and Margalit [15] give efficient algorithms for shortest paths on digraphs with edge weights from $\{-1, 0, +1\}$ costing $\widetilde{O}(n^\nu)$ where $\nu = \frac{3+\omega}{2}$. See also

Takaoka [16]. A part of Alon, et al.'s algorithm finds zero length directed paths and uses them as short-cuts. It may be possible to extract their 0 length path algorithm for short-cuts as the basis of our work. Nonetheless, Alon, et al.'s directed graph shortest-path algorithm takes $\widetilde{O}(n^{2.687})$ time when $\omega < 2.373$.

Galil and Margalit [17] extend the results of Alon, et al. [15] and integrate the shortest path distance and shortest path problem. Zwick [18] gives more efficient all-pairs shortest path and path distance algorithms. Zwick's shortest path's cost is better than $\widetilde{O}(n^{2.575})$, since $\omega < 2.373$. Our algorithm does not solve shortest-path problems.

Building on Barrett, et al., Bradford and Thomas [19] give a more efficient context free label constrained shortest path algorithm for graphs with positive and negative edge weights whose unlabeled versions have no negative cycles. Barrett, et al.'s algorithm for finding context free label constrained shortest paths with positive and negative edge weights costs $O(n^5|N|^2|R|^2)$. Bradford [20] gives a solution to a quickest-path problem for context-free grammars applied to cryptographic routing. Bradford and Choppella [21] use a special-case of Nykänen and Ukkonen's sign-closure algorithm for DAGs with initial edge costs from $\{-1, +1\}$, see also Khamespanah, Khosravi, and Sirjani [22]. Further Bradford and Choppella [21] find actual minimum-cost point-to-point Dyck paths in DAGs. Ward, Wiegand, and Bradford [23] give a distributed context-free labeled graph shortest path algorithm also based on [4]. Ward and Wiegand [24] analyze the complexity of wireless routing metrics as labeled path problems.

Chaudhuri [25] gives an $O(n^3/\log n)$ algorithm for context-free language reachability using an important dynamic-programming speedup method by Rytter [26].

After preprocessing a bidirected tree, Yuan and Eugster [27] give a $O(|V|\log|V|)$ algorithm for finding Dyck reachability in bidirected trees in $O(1)$ per query.

Zhang, Lyu, Yuan, Hao and Su [28] improve on Yuan and Eugster's bidirected tree algorithm. Zhang, et al. [28] also give an $O(|V|+|E|\log|E|)$ algorithm for determining Dyck reachability for bidirected digraphs. Each Dyck labeled edge in a bidirected digraph has a mirror edge going in the opposite direction and with a complimentary label. See also [29, 30].

Khamespanah, Khosravi, and Sirjani [22] use Nykänen and Ukkonen [2]'s exact path algorithm to improve their model checking algorithms for timed actors in distributed systems. They apply the pseudo-polynomial algorithm's $O(n^2)$ path relaxation cost, while accepting the pre-processing costs of $O(n^3)$. Our results break through this $O(n^3)$ barrier giving a $\widetilde{O}(n^\omega)$ algorithm. In general, Melski and Reps [14] discuss the "$O(n^3)$ bottleneck" for context-free program analysis. Our results solidly break through this bottleneck for the Dyck and semi-Dyck cases.

Dyck and semi-Dyck languages are also applied to data streaming, see Chakrabarti, Cormode, Kondapally and McGregor [31]. In addition, Tang, et al. [32] apply Dyck-CLF reachability to library summarization. Likewise, there are applications to database path queries, see Grahne, Thomo and Wadge [33]. Choppella and Haynes give an equivalence between unification graphs and Dyck path reachability problems in digraphs [34].

2 Dyck Path Reachability Problem

Direct application of standard shortest path [35] and transitive closure [35, 36] algorithms to LDGs does not seem to determine 0 reachability. In our instantiation of this challenge, such shortest paths use -1 edge weights or label-costs. That is, some paths may be negative. Indeed, shortest path algorithms gravitate towards negative paths. For this reason, intuitively shortest path algorithms not directly applicable.

This paper converts its edge labels to label-costs from $\{-1, 0, +1\}$. Therefore, rather than referring to labeled-costs, this paper just discusses edge costs. These costs are generally restricted to $\{-1, 0, +1\}$.

Next are definitions for sign-closure graphs from Nykänen and Ukkonen [2]. Define [2] the function $\mathbf{sgn}$, for $w \in \{-1, 0, +1\}$ so that $\mathbf{sgn}(w) = w$. For any LDG $G = (\Sigma, V, E_1)$, let M be a label-cost bound,

$$M = \max\{\, |\boldsymbol{lc}(e)| : e \in E_1 \,\}.$$

Throughout this paper, $M = 1$.

Definition 3 (Nykänen and Ukkonen [2]) Consider a digraph $G = (V, E)$. The *sign-closure* of $E(G)$ is $\mathbf{unsign}(G)$ which starts with $E(\mathbf{unsign}(G)) \leftarrow E(G)$, and then apply the rule:

> **if** $i \xrightarrow{v} k \xrightarrow{w} j \in E(\mathbf{unsign}(G))$ and $\mathbf{sgn}(v) \neq \mathbf{sgn}(w)$
> **then** put $i \xrightarrow{v+w} j$ in $E(\mathbf{unsign}(G))$,

until it no longer applies.

Applying Nykänen and Ukkonen's sign-closure algorithm finds semi-Dyck paths in an LDG. This relates semi-Dyck paths to transitive closure.

Changing the if-statement in Definition 3 as follows gives *Dyck* sign-closure. Given a LDG G, its Dyck sign-closure is $\mathbf{unsign}^{\geq}(G)$. To get the Dyck sign closure of a graph G, apply the rule

> **if** $i \xrightarrow{v} k \xrightarrow{w} j \in E(\mathbf{unsign}^{\geq}(G))$ and $\mathbf{sgn}(v) \neq \mathbf{sgn}(w)$ and $v \neq -1$
> **then** put $i \xrightarrow{v+w} j$ in $E(\mathbf{unsign}^{\geq}(G))$,

until it no longer applies.

Nykänen and Ukkonen show the sign-closure graph problem is $\mathcal{NP}$-Complete. Nonetheless, Nykänen and Ukkonen give a $O(M^2 n^3)$ time pseudo-polynomial algorithm for computing a sign-closure graph. This pseudo-polynomial algorithm runs in polynomial time for edge costs restricted to $\{-1, 0, +1\}$ since $M = 1$. In particular, when $M = 1$, computing a sign-closure graph costs $O(n^3)$ by [2]. We improve the cost to $\widetilde{O}(n^\omega)$.

The basic result of the next lemma is mentioned in the proof of Theorem 5 in Nykänen and Ukkonen [2]. Their Theorem 5 assumes their $O(M^2n^3)$ sign-closure algorithm. Nykänen and Ukkonen were not discussing Dyck or semi-Dyck languages, but in our context, their result is as follows.

Lemma 1 (Nykänen and Ukkonen [2]) Consider an LDG $G = (\Sigma, V, E_1)$ where Σ is Dyck (semi-Dyck) and $|\Sigma| = 2$. In computing a sign-closure with edge costs from $\{-1, +1\}$, then new edges added to $E(H)$ may be limited to costs from $\{ -1, 0, +1 \}$.

The case of Dyck languages follows since Dyck languages are also semi-Dyck languages. Given an LDG G, a 0 *cost* edge (path) is an edge (path) in $\mathbf{unsign}(G)$. A 0 cost edge has label-cost computed to be 0 and a 0 cost path has total cost 0.

Zero cost paths are semi-Dyck paths in G. A proof of the next lemma follows since semi-Dyck paths along ± 1 edges have equal numbers of $+1$ and -1 values. See also [21].

Lemma 2 The LDG $G = (\Sigma, V, E_1)$ where Σ is semi-Dyck and $|\Sigma| = 2$, then G has a semi-Dyck path between i and j iff in G there is a 0 cost path between i and j.

The next definition is well-known.

Definition 4 (Non-negative prefix sum) Suppose G is a LDG with a simple weighted path p from i_0 to i_{t+1}:

$$p \quad = \quad i_0 \xrightarrow{v_1} i_1 \xrightarrow{v_2} \cdots \xrightarrow{v_t} i_t \xrightarrow{v_{t+1}} i_{t+1}.$$

then node i_k has prefix sum $v_1 + \cdots + v_k$ from $p = i_0$ to i_k along p for $k : t+1 \geq k \geq 1$. The prefix sum for i_0 is 0. In a path p, if p's prefix sums for all 0 cost subpaths are non-negative, then the path p has a non-negative prefix sum.

A proof of the next lemma follows a proof of Lemma 2, see also [21, 8].

Lemma 3 The LDG $G = (\Sigma, V, E_1)$ where Σ is Dyck and $|\Sigma| = 2$, then, G has a Dyck path between i and j iff in G there is a 0 cost path between i and j having only non-negative prefix sums.

The next lemma includes labeled edges going from a node to itself. This paper assumes no self-cycles with repeated edges.

Lemma 4 Consider an LDG $G = (\Sigma, V, E_1)$ where Σ is Dyck (semi-Dyck), $|\Sigma| = 2$ with sign-closure $\mathbf{unsign}^{\geq}(G)$ ($\mathbf{unsign}(G)$). Then all vertices in V have at most $3n$ outgoing edges.

3 Efficient exact $-1, 0$, and $+1$ paths

This section shows how to determine which nodes have *exact* paths of costs $\{-1, 0, +1\}$ in LDGs with $\{-1, +1\}$ weighted edges in $\widetilde{O}(n^\omega)$ operations. Our new solution is expressed as Dyck or semi-Dyck paths in LDGs. This is done by computing sign-closures of digraphs with $\{-1, +1\}$ edge weights. In the process, 0 cost edges may be added to these diagraphs. Also, ± 1 and 0 edges are extended by 0 cost edges. Our algorithm uses algebraic matrix multiplication of specially coded matrices. These matrix encodings are from Alon, Galil, and Margalit [15]. See also Yuval [37]. Each algebraic matrix multiplication may be done in $O(n^\omega \log n)$. This may be improved by a polylog factor, see for example [15, 38, 25, 18, 26].

Alon, Galil, and Margalit's shortest path algorithm [15] starts by finding exact 0 length paths in digraphs with edge costs $\{-1, 0, +1\}$. They use these 0 exact paths as shortcuts to find shortest paths. Our algorithm finds $\{-1, 0, +1\}$ exact paths between all pairs of vertices. Alon, et al.'s digraph shortest path algorithm works for edges with much larger costs. Their digraph shortest path algorithm is substantially more costly than our digraph exact path algorithms. Of course, they solve the shortest path algorithm where we solve a reachability problem.

Matrices are written in uppercase and their elements are written in lowercase [18]. Matrix parenthesized superscripts, such as those in $\{D^{(-1)}, D^{(0)}, D^{(+1)}\}$ signify different matrices. These parenthesized powers are not exponentiation. Likewise, matrix elements raised to powers, such as $d_{i,j}^{-1}, d_{i,j}^{0}, d_{i,j}^{+1}$, are not exponentiated. Rather these superscripts indicate the matrices these elements are from. In this case, $d_{i,j}^{-1}$ is in $D^{(-1)}$, $d_{i,j}^{0}$ is in $D^{(0)}$, and $d_{i,j}^{+1}$ is in $D^{(+1)}$.

The algorithm in Figure 1 maintains three adjacency matrices $D^{(-1)}, D^{(0)}$, and $D^{(+1)}$. These three adjacency matrices allow $-1, 0$ and $+1$ edges to go from any vertex to any other vertex.

Given an LDG $G = (\Sigma, V, E_1)$, where $|\Sigma| = 2$, define the adjacency matrices $D^{(g)} \in \{D^{(-1)}, D^{(0)}, D^{(+1)}\}$ whose edge costs are from $\{-1, 0, +1\}$. Before iteration $\ell = 1$, there are no 0 cost edges in $D^{(0)}$.

$$d_{i,j}^{g} \quad = \quad \begin{cases} -1 & \text{if } (i,j) \in E_\ell \text{ and } g = -1 = \boldsymbol{lc}(i,j) \\ 0 & \text{if } (i,j) \in E_\ell \text{ and } g = 0 = \boldsymbol{lc}(i,j) \text{ and } \ell > 1 \\ +1 & \text{if } (i,j) \in E_\ell \text{ and } g = +1 = \boldsymbol{lc}(i,j) \\ \infty & \text{otherwise.} \end{cases}$$

Subsequently, in each iteration a new edge set E_ℓ is created in the ℓ-th iteration of our main algorithm. At this point, any new $\{-1, 0, +1\}$ cost paths are placed in E_ℓ during iteration ℓ.

So, during computation there may be at most three (different) labeled edges directly from any vertex i to any other vertex j. See Lemma 4. Recall, the initial graph edges only have weights from $\{-1, +1\}$. The algorithm in Figure 1 implements these equations to find all $-1, 0, +1$ exact paths. This algorithm is substantially less efficient than Nykänen and Ukkonen [2] applied to graphs with $\{-1, +1\}$ edges. However, Figure 1's algorithm forms a basis for our more efficient algorithm.

```
**Expensive-Digraph-exact-paths**: $\pm 1$, for the semi-Dyck LDG $G = (V, E_1)$
1. $\{ D^{(-1)}, D^{(0)}, D^{(+1)} \} \leftarrow$ **Init-Adjacency-Matrices**$(G)$
2. $n \leftarrow |V|$
3. for $\ell \leftarrow 2$ to $n$ do
4.   for $i \leftarrow 1$ to $n$ do
5.     for $j \leftarrow 1$ to $n$ do
6.       for $k \leftarrow 1$ to $n$ do
7.         if $(d^{-1}_{i,k} + d^{+1}_{k,j}) = 0 \vee (d^{+1}_{i,k} + d^{-1}_{k,j}) = 0$ then $d^{0}_{i,j} \leftarrow 0$
8.         if $(d^{+1}_{i,k} + d^{0}_{k,j}) = 1 \vee (d^{0}_{i,k} + d^{+1}_{k,j}) = 1$ then $d^{+1}_{i,j} \leftarrow +1$
9.         if $(d^{-1}_{i,k} + d^{0}_{k,j}) = -1 \vee (d^{0}_{i,k} + d^{-1}_{k,j}) = -1$ then $d^{-1}_{i,j} \leftarrow -1$
```

Figure 1: An inefficient $-1, 0, +1$ exact path algorithm for digraphs with initial edge costs $\{ -1, +1 \}$

The function **Init-Adjacency-Matrices** in Figure 1 initializes each of $D^{(-1)}, D^{(0)}, D^{(+1)}$ with sufficiently large values representing no edge and no path. No path and no edge in the algorithm in Figure 1 may be represented by numbers as little as 2. Although, if there is no t to get from i to j, then $d^{t}_{i,j}$ is effectively infinite, for any $t \in \{-1, 0, +1\}$. Next **Init-Adjacency-Matrices** represents a -1 edge from i to j in $D^{(-1)}$ by placing -1 in $d^{-1}_{i,j}$. Likewise $+1$ edges are represented in $D^{(+1)}$ by appropriate placement of $+1$ values.

Definition 5 (E_1-length) Consider an LDG $G = (V, E_1)$ and an exact path p in G. The E_1-*length* of p is the number of E_1 edges in p.

Exact 0 cost paths have even E_1-lengths. Exact ± 1 cost paths have odd E_1-lengths.

The algorithm in Figure 1 may be made a little more efficient. In the next section of this paper, we give a substantially more efficient solution building on this approach.

Lemma 5 Consider an LDG $G = (\Sigma, V, E_1)$, where Σ is semi-Dyck with $|\Sigma| = 2$. Then at the termination of Figure 1's algorithm, $d^{w}_{i,j} = w$, for $w \in \{-1, 0, +1\}$, for all $\{i, j\} \subseteq V$ where there is an exact w cost path from i to j.

Proof: This is shown by complete induction on the iteration ℓ for exact $-1, 0$ and $+1$ paths.

Basis Immediately after the initial iteration $\ell = 2$, all 0 exact paths of even E_1-length at least 2 are found by line 7. Such exact 0 paths are created by combining adjoining $+1$ and -1 edges or combining adjoining -1 and $+1$ edges. An exact 0 path from i to j is recorded in the matrix $D^{(0)}$ by setting $d^0_{i,j}$ to 0.

In iteration $\ell = 3$ the exact 0 cost paths from iteration $\ell = 2$ are combined with adjoining ± 1 edges from E_1 giving ± 1 exact paths. These exact ± 1 paths have odd E_1-length of at least 3. This is done by lines 8 and 9 and the matrices $D^{(\pm 1)}$ record these exact paths.

Inductive Hypothesis For $\ell = 2$ and $\ell = 3$, the next cases hold.

Immediately after iteration $\ell : \ell \geq t \geq 1$, for all even t, this algorithm finds all exact 0 cost paths in line 7. By assumption, for all even t, these new exact 0 paths discovered in iteration t have even E_1-length of at least t. Line 7 combines adjoining ± 1 and ∓ 1 exact paths from previous iterations and records these paths in $D^{(0)}$. These new exact 0 paths are recorded in $D^{(0)}$.

After iteration $\ell : \ell \geq t \geq 1$, for all odd t, this algorithm finds all exact ± 1 cost paths in lines 8 and 9. By assumption, for all odd t, these new exact ± 1 paths discovered in iteration t have odd E_1-length of at least t. Lines 8 and 9 combines exact 0 paths and exact ± 1 paths from previous iterations. These new exact ± 1 paths are recorded in $D^{(\pm 1)}$.

Inductive Step Consider the algorithm immediately after iteration $\ell + 1$ where $\ell + 1$ is even. By the inductive hypothesis, consider all odd $t : \ell \geq t \geq 1$, the matrices $D^{(\pm 1)}$ contain ± 1 exact paths of odd E_1-length. Also, all ∓ 1 exact paths in $D^{(\mp 1)}$ are of odd E_1-length. In iteration $\ell + 1$, this algorithm combines adjoining ± 1 and ∓ 1 exact paths to form 0 exact paths of even E_1-length. Suppose an exact 0 cost path p is discovered in iteration $\ell + 1$ where p is of E_1-length $t \leq \ell - 1$. This cannot be the case since by the inductive hypothesis p would have been discovered in iteration t.

Consider the algorithm immediately after iteration $\ell + 1$ where $\ell + 1$ is odd. By the inductive hypothesis for all even $t : \ell \geq t \geq 1$, the matrices $D^{(0)}$ contain exact 0 paths of even E_1-length. Likewise, for odd $t : \ell \geq t \geq 1$, the matrices $D^{(\pm 1)}$ contain ± 1 exact ± 1 paths of odd E_1-length. In this case, the algorithm combines adjoining ± 1 (0) exact paths with 0 (± 1) exact paths giving new exact ± 1 cost paths of odd E_1-length. Suppose an exact ± 1 cost path p is discovered in iteration $\ell + 1$ where p is of E_1-length $t \leq \ell - 1$. This cannot be the case since by the inductive hypothesis p would have been discovered by iteration t. ∎

Lemma 6 Consider an LDG $G = (\Sigma, V, E_1)$, where Σ is semi-Dyck with $|\Sigma| = 2$, and the algorithm in Figure 1. At the termination of the algorithm, if $d^w_{i,j} = w \in \{-1, 0, +1\}$, for any $\{i, j\} \subseteq V$ where there is a w cost exact path from i to j, then the algorithm computed the sign-closure $\mathbf{unsign}(G)$.

Proof: At the termination of the algorithm $d^w_{i,j} = w$, where $w \in \{-1, 0, +1\}$, for all $\{i, j\} \subseteq V$ where there is an exact w cost path from i to j. If $\mathbf{unsign}(G)$ is not complete, then some edge from i to j must not have been placed in $E(\mathbf{unsign}(G))$, by applying the sign-closure rule

if $i \xrightarrow{v} k \xrightarrow{w} j \in E(\mathbf{unsign}(G))$ and $\mathbf{sgn}(v) \neq \mathbf{sgn}(w)$
then put $i \xrightarrow{v+w} j$ in $E(\mathbf{unsign}(G))$.

Since $v \neq w$, then there must be some path from i to j that was not generated by the algorithm in Figure 1.

But, $i \xrightarrow{v+w} j$ is an exact $v + w \in \{-1, 0, +1\}$ path from i to j. This exact path must have been found by Lemma 5, completing the proof. ▮

Nykänen and Ukkonen's sign-closure algorithm [2] finds exact paths in graphs in $O(n^3)$ time. The next result shows how to find Dyck exact $-1, 0$, and $+1$ paths. This is done by dropping the condition $d^{-1}_{i,k} + d^{+1}_{k,j}$ from the calculation of $d^0_{i,j}$ in Figure 3.

Lemma 7 Consider an LDG $G = (\Sigma, V, E_1)$, where Σ is Dyck with $|\Sigma| = 2$, and the algorithm in Figure 1 while dropping the expression $(d^{-1}_{i,k} + d^{+1}_{k,j}) = 0$ in line 7. At the termination of the algorithm, if $d^w_{i,j} = w \in \{-1, 0, +1\}$, for any $\{i, j\} \subseteq V$ where there is a w cost Dyck path from i to j, then the algorithm computed the Dyck sign-closure $\mathbf{unsign}^{\geq}(G)$.

Proof: Lemma 5 shows this algorithm finds all $-1, 0$, and $+1$ exact paths for any semi-Dyck LDG G. With this in mind, it remains to extend that lemma. The next arguments allows the extension of Lemma 5's induction proof to this Dyck case.

Lines 7, 8, and 9 in Figure 1, only computing $d^0_{i,j}$ may create a negative prefix sum for a 0 cost path or subpath. Clearly computing $d^{+1}_{i,j}$ cannot have a negative prefix sum. Likewise, computing $d^{-1}_{i,j}$ can't compute a negative prefix sum for a 0 cost path or subpath. Computing $d^{-1}_{i,j}$ just extends -1 edges, but does not have non-Dyck labeled paths.

Removing the condition $(d^{-1}_{i,k} + d^{+1}_{k,j}) = 0$ in line 7 in the equation for $d^w_{i,j}$ finds all exact $w \in \{-1, 0, +1\}$ cost paths without negative prefix sums. An identical argument as in the proof of Lemma 6 indicates all edges of $\mathbf{unsign}^{\geq}(G)$ have been found. Thus, this modification computes the Dyck sign-closure. ▮

Intuitively, our approach to improving the algorithm in Figure 1 is anchored in Boolean matrix multiplication for transitive closure. However, starting with graph edges $\{-1, +1\}$ and computing with the edge weights $\{-1, 0, +1\}$ seems to preclude Boolean matrix multiplication. Thus, we leverage Alon, Galil, and Margalit [15].

3.1 AGMY matrix encoding

Alon, Galil, and Margalit [15] as well as Yuval [37] supply the basis of our *(AGMY)* algebraic matrix coding. These AGMY style codings have been very fruitful, see for example [18, 39, 40, 16, 17].

Lemma 4 gives insight into an algebraic matrix product solution. In particular, the AGMY representation uses powers of $3(n+1)$ to differentiate $\{-1, 0, +1\}$ edge weights. That is, $\frac{1}{3(n+1)}, (3(n+1))^0, 3(n+1)$ represent $-1, 0, +1$ edges, respectively. These AGMY values are sufficiently separated to allow information to be gleaned after an algebraic matrix product.

Figure 2 shows how to translate adjacency matrices $D^{(-1)}, D^{(0)}$ and $D^{(+1)}$ to an AGMY encoded adjacency matrix. The restriction $g \neq h$ is from the sign-closure in Definition 3.

$$c_{i,j} \leftarrow \begin{cases} \displaystyle\sum_{\substack{k=1 \\ g \neq h}}^{n} (3(n+1))^{d^g_{i,k} + d^h_{k,j}} & \text{if } d^g_{i,k} \neq \infty \wedge d^h_{k,j} \neq \infty \\ 0 & \text{if } d^g_{i,k} = \infty \vee d^h_{k,j} = \infty. \end{cases}$$

Figure 2: AGMY matrix coding for algebraic matrix multiplication to simuate one matrix dot product based on Alon, Galil, and Margalit [15]; and Yuval [37]

An algebraic matrix product computes the expression in Figure 2 for all $i, j : n \geq i, j \geq 1$, see Figure 3. If $d^g_{i,k} = \infty$ or $d^h_{k,j} = \infty$, then replace $(3(n+1))^{d^g_{i,k} + d^h_{k,j}}$ with 0. This works since no finite power of $3(n+1)$ is 0.

For all $i, j : n \geq i, j \geq 1$, let

$$\begin{aligned} c^{-1}_{i,j} &\leftarrow (3(n+1))^{d^0_{i,k} + d^{-1}_{k,j}} + (3(n+1))^{d^{-1}_{i,k} + d^0_{k,j}} \\ c^{0}_{i,j} &\leftarrow (3(n+1))^{d^{-1}_{i,k} + d^{+1}_{k,j}} + (3(n+1))^{d^{+1}_{i,k} + d^{-1}_{k,j}} \\ c^{+1}_{i,j} &\leftarrow (3(n+1))^{d^0_{i,k} + d^{+1}_{k,j}} + (3(n+1))^{d^{+1}_{i,k} + d^0_{k,j}} \end{aligned}$$

Figure 3: A breakout of computing AGMY matrix values

Initially, an adjacency matrix represents an LDG G with edge costs from $\{-1, +1\}$. So at the start of the algorithm, any two vertices i and j may share a -1 and $+1$ edge going in each direction. Thus, each element of the initial AGMY coded adjacency matrices starts with values from,

$$\{\, 0,\ (3(n+1))^1,\ (3(n+1))^{-1},\ (3(n+1))^1 + (3(n+1))^{-1} \,\}.$$

Here the AGMY 0 represents no edge.

During the first matrix product, 0 cost edges may appear. They are represented by $(3(n+1))^0 = 1$.

The idea for the next Lemma are based on Alon, Galil, and Margalit [15].

Lemma 8 Given two $n \times n$ AGMY encoded LDG adjacency matrices S and T *representing* values from $\{-1, 0, +1\}$. Consider an algebraic matrix product $P = ST$ and say there is a path of cost $p_{i,j}$ between i and j, then

$$n\left[(3(n+1))^2 + 2(3(n+1)) + 3 + \frac{2}{3(n+1)} + \frac{1}{(3(n+1))^2}\right] \ \geq \ p_{i,j}.$$

Proof: While a sign-closure is computed, any two vertices i and j may share up to three edges going in each direction by Lemma 4. In AGMY coding, three outgoing edges are bounded by,

$$B \ \leq \ \frac{1}{3(n+1)} + 1 + 3(n+1).$$

Thus, a single algebraic matrix product produces a new matrix element of at most

$$(3(n+1))^2 + 2(3(n+1)) + 3 + \frac{2}{3(n+1)} + \frac{1}{(3(n+1))^2} \ \geq \ B^2. \qquad (1)$$

The dot-product of row $S[i,*]$ and column $T[*,j]$ gives a value of at most,

$$\underbrace{B^2 + \cdots + B^2}_{\text{the sum of } n \text{ squares}}$$

and since there are at most n of these B^2 terms, the result holds. ∎

Following the AGMY adjacency matrices of Lemma 8, Say there is a path from i to j, then the dot-product of row $S[i,*]$ and column $T[*,j]$ is at least AGMY $\frac{1}{3(n+1)}$. This is the result of combining adjoining -1 and 0 edges. This is because a -1 cost edge is represented by AGMY $\frac{1}{3(n+1)}$ and a 0 edge is represented by AGMY $(3(n+1))^0 = 1$.

Factors besides $\{\frac{1}{3(n+1)}, 1, 3(n+1)\}$ in Lemma 8 are removed after each matrix product. These factors represent unnecessary intermediary paths and their growth makes the algorithm too expensive. So, following Alon, et al. [15], see also Zwick [18], our algorithm removes all edges except for AGMY $\{\frac{1}{3(n+1)}, 1, 3(n+1)\}$. This is normalization. Normalization removes unnecessary edges for computing the EPL for 0 cost paths. So, immediately after the matrix product in line 5, the adjacency elements are converted back to values from $\{\frac{1}{3(n+1)}, 1, 3(n+1)\}$.

The next corollary follows from the upper bound on the representation of each adjacency element from Lemma 8. Very similar results are in [15, 18].

Corollary 1 In a single AGMY algebraic matrix multiplication of an LDG's adjacency matrix, each of the resulting matrix elements may be represented in $O(\log n)$ bits.

New ± 1 edges are generated by matrix products and normalization or extending ± 1 by 0 cost edges. Furthermore, in computing Dyck paths, any edge $i \xrightarrow{-1} k$ edge may *not* be joined to $k \xrightarrow{+1} j$. This is because

$$i \xrightarrow{-1} k \xrightarrow{+1} j$$

is not Dyck.

In the case when a -1 edge is made from multiple E_1 edges, then this edge represents a path. Therefore, this path has a negative sum. In fact, it is a negative prefix sum. Such a -1 edge is already not Dyck.

```
1. $\textit{detectNegativeOneEdge}(\text{edge\_cost}, n)$
2.   check $\leftarrow 3(n+1) \times$ fractional_part(edge_cost)
3.   **if** $2n \geq$ check $\geq 1$ **then return** True
4.   **else return** False

1. $\textit{detectPositiveOneEdge}(\text{edge\_cost}, n)$
2.   check $\leftarrow$ truncate(edge_cost$/3(n+1)$)
3.   **if** $2n \geq$ check $\geq 1$ **then return** True
4.   **else return** False

1. $\textit{detectZeroEdge}(\text{edge\_cost}, n)$
2.   check $\leftarrow$ truncate(edge_cost) **mod** $3(n+1)$
3.   **if** $3n \geq$ check $> 0$ **then return** True
4.   **else return** False
```

Figure 4: Functions used to detect AGMY costs representing $\{-1, 0, +1\}$ for normalize an AGMY algebraic matrix product

Normalization uses the functions in Figure 4. The upper and lower bounds in each function in Figure 4 are determined as follows. Lemma 8 gives upper bounds for

```
Digraph-flat-exact-paths($G$)
// $G = (\Sigma, V, E_1)$ is an LDG, $\Sigma$ is semi-Dyck and $|\Sigma| = 2$
1. $\{D^{(-1)}, D^{(0)}, D^{(+1)}\} \leftarrow$ Init-Adjacency-Matrices($G$)
2. $M \leftarrow$ AGMY-Code-then-Sum($D^{(-1)}, D^{(0)}, D^{(+1)}$)
3. $n \leftarrow |V|$
4. for $\ell \leftarrow 2$ to $\lceil \log n \rceil + 1$ do
5.     $M \leftarrow M\,M$  // AGMY, find new 0 edges
6.     Remove $\pm 1$ edges from $M$
7.     $M \leftarrow$ Normalize_and_Divide_by_2($M$)
8.     $Z \leftarrow$ Get-Zero-Edges($M$)
9.     $M \leftarrow Z\,M\,Z$  // AGMY, extend $\pm 1$ and 0 edges
```

Figure 5: A new matrix-based sign-closure algorithm for digraphs with initial edge costs representing $\{-1, +1\}$. This algorithm uses AGMY coded matrices.

detectNegativeOneEdge(edge_cost, n). That is, the first operation of *detectNegativeOneEdge* is to multiply the fractional part of its edge-weight by $3(n+1)$ giving the upper-bound

$$2n \quad = \quad 3(n+1)\left(\frac{2n}{3(n+1)}\right).$$

Since an AGMY $\frac{2}{3(n+1)}$ term in Equation 1 indicates there are at most two different ways to form a -1 edge edge through a single intermediary vertex. For example, take paths from a vertex i to another vertex j with intermediary k. That is, the two ways are: $i \xrightarrow{-1} k \xrightarrow{0} j$ or $i \xrightarrow{0} k \xrightarrow{-1} j$. Equation 1 also indicates there is at most one cost -2 edge between any two vertices due to the $\frac{1}{(3(n+1))^2}$ term. A potential -2 cost edge will not be detected by *detectNegativeOneEdge* because a single -1 edge that has AGMY cost of at least $\frac{1}{3(n+1)}$. Thus, the first operation of *detectNegativeOneEdge* is to multiply the fractional part of the AGMY edge-weight by $3(n+1)$ so the smallest value of a -1 cost edge is 1. The upper and lower bounds in *detectPositiveOneEdge* and *detectZeroEdge* are similar.

Thus, reset all elements immediately after each recursive doubling step using the functions in Figure 4. The three functions in Figure 4 detect $-1, 0$ and $+1$ cost edges following each algebraic AGMY matrix multiplication. These AGMY edges have values as large and complex as those in Lemma 8. After detecting $-1, 0$ or $+1$ AGMY edges, more complex pathways are simplified or normalized by replacing them by the appropriate members of $\{\frac{1}{3(n+1)}, 1, 3(n+1)\}$. Each normalization costs $O(n^2 \log n)$.

Figure 5 is the critical component of all our results. In Figure 5, *Normalize_and_Divide_by_2* removes redundant edges. By Lemma 8, in line 5 the algebraic AGMY matrix multiplication gives values as large as

$$n\left[(3(n+1))^2 + 2(3(n+1)) + 3 + \frac{2}{3(n+1)} + \frac{1}{(3(n+1))^2}\right].$$

Line 6 removes ± 1 edges. In line 7, normalization changes ± 2 cost edges to ± 1 cost edges. Thus, only retaining AGMY encodings for $\{\frac{1}{3(n+1)}, 1, 3(n+1)\}$. The idea of dividing the edge costs by 2 is from Alon, et al. [15].

Line 9 joins adjacent 0 cost edges and it extends ± 1 cost edges with adjoining 0 cost edges. Line 9 cannot generate ± 2 cost edges. Before line 9 in iteration ℓ, these adjoining 0 cost edges have E_1-length from 2 up to $3^{\ell-2} \cdot 2^{\ell-1}$. So, at the end of line 9 in iteration ℓ, up to three consecutive adjoining 0 cost edges may form a single E_1-length $3^{\ell-1} \cdot 2^{\ell-1}$ edge.

```
5'.  $M' \leftarrow Markup\_minus\_one\_edges(M)$  // AGMY, mark $-1$ edges
5''. $M \leftarrow M' M$  // AGMY product, non-Dyck 0 edges are detectable
```

Figure 6: Replacing line 5 of Algorithm 5 with these lines is the basis of a sign-closure algorithm for Dyck digraphs with initial edge costs $\{-1, +1\}$. Further updates are done in *Normalize_and_Divide_by_2*(M) so it deletes 0 edges marked here as created by non Dyck paths.

Figure 6 shows how to determine if a new 0 edges is made by a -1 edge followed by a $+1$ edge. A -1 edge followed by a $+1$ edge is not Dyck. The function *Markup_minus_one_edges* adds $\left(\frac{1}{3(n+1)}\right)^4$ to each negative edge in its AGMY input matrix M producing M'. Now, a -1 cost edge followed by a $+1$ cost edge has an AGMY product of

$$\left(\left(\frac{1}{3(n+1)}\right)^4 + \left(\frac{1}{3(n+1)}\right)\right) 3(n+1) \quad = \quad \left(\frac{1}{3(n+1)}\right)^3 + 1$$

and such cubic terms are found by a Dyck modified *Normalize_and_Divide_by_2*(M) in the algorithm in Figure 5. This cubic term only occurs when an AGMY -1 value is multiplied by an AGMY $+1$ value, when this product represents a -1 edge going to a $+1$ edge. Of course, a -1 edge going to a $+1$ edge is not a Dyck path. Also these cubic terms cannot become -2 AGMY edges in one AGMY matrix multiplication, by Lemma 8. Finally, for the Dyck case, the function *Normalize_and_Divide_by_2*(M) in the algorithm in Figure 6 is updated to delete any 0 cost edge made from a -1 edge followed by a $+1$ edge.

The next convention smooths the subsequent presentation.

Convention 1 (Digraph-flat-exact-paths context) The algorithm **Digraph-flat-exact-paths** refers to Figure 5 for the semi-Dyck case in addition to Figure 6 for the Dyck case, depending on the context.

3.2 Flat Dyck and semi-Dyck grammars

Flat grammars supply a foundation for the complete solution.

Definition 6 (Flat Dyck and flat semi-Dyck grammars) The flat Dyck language is,

$$\begin{aligned} F &\implies T \mid P \\ T &\implies T\,T \mid P \mid \epsilon \\ P &\implies \mathbf{a}\,P\,\mathbf{a}^{-1} \mid \epsilon. \end{aligned}$$

The flat semi-Dyck language replaces the last production with $P \implies \mathbf{a}\,P\,\mathbf{a}^{-1} \mid \mathbf{a}^{-1}\,P\,\mathbf{a} \mid \epsilon$.

Definition 7 (Sign-closure graph evolution) Consider an LDG $G_1 = (\Sigma, V, E_1)$ and **Digraph-flat-exact-paths**. At the end of each iteration $\ell \geq 2$ this algorithm produces the LDG $G_\ell = (\Sigma \cup \{0\}, V, E_\ell)$.

A path in G may become a single $\{-1, 0, +1\}$ edge in E_ℓ. Such new edges are generated by **Digraph-flat-exact-paths**. The number of $+1$ and -1 edges from E_1 contributing to new edges give important insights.

Definition 8 (c_+-length and c_--length) Consider an LDG $G_\ell = (\Sigma, V, E_\ell)$ and an edge $e \in E_\ell$, for $\ell \geq 1$. An edge e's $c_+(e)$-length is e's number of $+1$ edges from E_1 and $c_-(e)$-length is e's number of -1 edges from E_1.

In iteration ℓ of **Digraph-flat-exact-paths**, edges with ±1 costs are made by joining ±1 edges from iteration $\ell - 1$. Also, additional exact 0 cost paths may be included in the new edges. After iteration $\ell = 2$, new ±1 edges are not ±1 exact paths.

The next corollary is known in a number of contexts.

Corollary 2 Consider an LDG $G_1 = (\Sigma, V, E_1)$ where $|\Sigma| = 2$, Σ is Dyck (semi-Dyck) and any 0 cost edge e in G_ℓ with E_1-length $|e|$, then e has $\frac{|e|}{2}$ edges with $+1$ costs and $\frac{|e|}{2}$ edges with -1 costs, all from E_1.

Corollary 2 indicates $c_+(e) = c_-(e) = \frac{|e|}{2}$ for all 0 cost edges e. The c_+-length (c_--length) may be any integer from 0 to $n - 1$.

The proof of the next lemma uses the idea that a new ±1 edge e may merge with an exact 0 cost edge producing a new ±1 edge e' with a larger E_1-length. However both edges e and e' have the same $c_\pm$-length.

Lemma 9 Given an LDG $G_1 = (\Sigma, V, E_1)$, where Σ is flat Dyck (flat semi-Dyck) and $|\Sigma| = 2$, then all exact 0 cost edges created by line 5 in iteration ℓ of **Digraph-flat-exact-paths** have E_1-length at least $2^{\ell-1}$, where $\lceil \log n \rceil + 1 \geq \ell \geq 2$.

Proof: Without loss, this proof focuses on the algorithm in Figure 5. Line 9 extends ± 1 and exact 0 cost edges using 0 edges of E_1-length from 2 to $3^{\ell-2} \cdot 2^{\ell-1}$. It extends adjoining pairs of exact 0 cost paths in flat grammars. Thus, line 9 is not in the next induction. The induction is on the iteration ℓ and includes $\{-1, 0, +1\}$ edges of E_1-length of at least $2^{\ell-1}$.

Basis In iteration $\ell = 2$, line 5 computes exact 0 cost edges of E_1-length at least $2^{\ell-1} = 2$. Likewise, line 5 generates all ± 2 cost edges with E_1-length of at least $2^{\ell-1} = 2$. These ± 2 cost edges are converted to ± 1 cost edges, by normalization in line 7. Their E_1-lengths remain at least $2^{\ell-1}$, but their c_+-length or c_--length is $2^{\ell-1}$.

Inductive Hypothesis Assume for some λ, all iterations ℓ where $\lambda \geq \ell \geq 2$ are such that line 5 computes $\{-1, 0, +1\}$ cost edges of E_1-length at least $2^{\ell-1}$. Here the 0 cost edges are exact 0 cost paths. The new ± 1 edges have c_+-length and c_--length of $2^{\ell-1}$, respectively.

Inductive Step Consider iteration $\lambda + 1$ for some λ where $\lambda \geq \ell \geq 2$.

By the inductive hypothesis, in iteration $\lambda = \ell$, line 5 computes exact 0 cost edges of E_1-length at least $2^{\lambda-1}$.

In iteration $\lambda = \ell$, *Normalize_and_Divide_by_2* produces new ± 1 edges only if they are ± 2 edges just generated by line 5. These new ± 1 edges have E_1 length of at least $2^{\lambda-1}$ by the inductive Hypothesis. Also the inductive Hypothesis indicates these new ± 1 edges have c_+-length or c_--length of $2^{\ell-1}$, respectively.

In conclusion, during iteration $\lambda + 1$, line 5 combines adjoining $+1$ and -1 cost edges forming exact 0 cost edges with E_1-length at least $2 \cdot 2^{\lambda-1} = 2^{\lambda}$. Also, the new $+1$ edges have c_+-length 2^{λ}, and the new -1 edges have c_--length 2^{λ}. This is because new ± 2 edges are made by joining ± 1 edges from the previous iteration. ▮

The algorithm **Digraph-flat-exact-paths** uses $O(\log n)$ algebraic matrix multiplications. The same is true for the Dyck extension given in Figure 6. Each matrix multiplication costs $O(n^{\omega} \log n)$. This gives a total cost of $O(n^{\omega} \log^2 n)$ time for the flat Dyck (semi-Dyck) case.

Theorem 1 Given an LDG $G = (\Sigma, V, E_1)$, where Σ is flat Dyck (flat semi-Dyck) and $|\Sigma| = 2$, then Figure 5's algorithm (updated by Figure 6) finds all exact 0 paths in $\widetilde{O}(n^{\omega})$ time.

4 Dyck and semi-Dyck grid graphs

Dyck and semi-Dyck languages are generalizations of the Flat Dyck and flat semi-Dyck languages. Grid paths enable the transition from flat grammars to the general case. Each acyclic path in an LDG has an equivalent path in a grid graph. See an example grid graph in Figure 7.

Pyramids and valleys are distinct grid graphs. Pyramid paths are generated by the non-terminal P in Definition 6. Pyramids have Dyck labels $\mathbf{a}^k\,\mathbf{a}^{-k}$, for $k \geq 1$. Two adjoint pyramids in a Dyck grid paths share a valley. This shared valley is not a Dyck path on its own. Indeed, pyramids are building blocks of Dyck paths in grid graphs. In semi-Dyck grid paths, the valleys are themselves semi-Dyck words. That is, pyramids and valleys are building blocks for semi-Dyck paths in grid graphs. Semi-Dyck path valleys are labeled $\mathbf{a}^{-k}\,\mathbf{a}^k$, for $k \geq 1$.

Definition 9 (Pyramids and Valleys) Consider an LDG $G = (\Sigma, V, E_1)$. A Dyck pyramid is a maximal path labeled by $\mathbf{a}^k\,\mathbf{a}^{-k}$, for $k \geq 1$. A Dyck pyramid path p is maximal since its label is $\mathbf{a}^k\,\mathbf{a}^{-k}$ and $\mathbf{a}^{k+1}\,\mathbf{a}^{-k-1}$ does not label a valid Dyck path containing p.

Semi-Dyck paths also includes maximal valleys labeled by $\mathbf{a}^{-k}\,\mathbf{a}^k$.

A *peak* is labeled $\mathbf{a}\,\mathbf{a}^{-1}$. In a grid graph, a pyramid starting from (i, j) and ending at $(i + t, j)$ has *base level* j. Figure 9 shows base levels $0, 1$ and 2. The y-axis of Figure 7 shows grid levels.

In a grid graph, a semi-Dyck path starts from point $(0, 0)$ and ends at some point $(x, 0)$ for an integer $x \geq 0$. A Dyck path in a grid graph never has an y coordinate below 0. In general, an LDG $+1$ edge is equivalent to a grid graph edge going from (x, y) to $(x + 1, y + 1)$, for $x \geq 0$. Likewise, an LDG -1 edge is equivalent to a grid graph edge going from (x, y) to $(x + 1, y - 1)$.

Let p be an exact 0 cost path in an LDG. In a grid graph p is,

$$p \;=\; (x_1, y_1),\, (x_2, y_2),\; \cdots\,,\; (x_n, y_n).$$

so that $(x_1, y_1) = (0, 0)$, $(x_n, y_n) = (x_n, 0)$, and $y_1 + \cdots + y_n = 0$. In such a path, its *maximal peak(s)* are at level $\max\{\, y_1,\; \cdots\,,\; y_n\,\}$.

Start with an exact 0 cost path p, then two exact 0 cost subpaths p_1 and p_2 are distinct iff $E[p_1] \cap E[p_2] = \emptyset$. Suppose p does not form a cycle. Two exact 0 cost distinct subpaths p_1 and p_2 are *adjoining* when they share exactly one vertex and have the same base level. This common vertex joins the end of one of these paths to the start of the other.

Definition 10 (Pairs) A *pyramid pair* is an adjoining pair of pyramids. A *valley pair* is an adjoining pair of valleys. Likewise, a *mixed pair* is an adjoining pyramid (valley) and valley (pyramid).

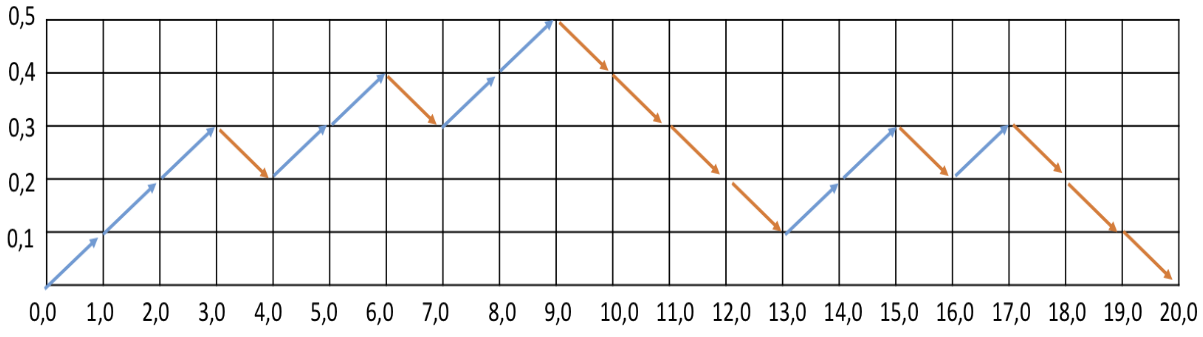

Figure 7: An exact 0 cost Dyck path on a grid with maximum peak $(9,5)$ at level 5

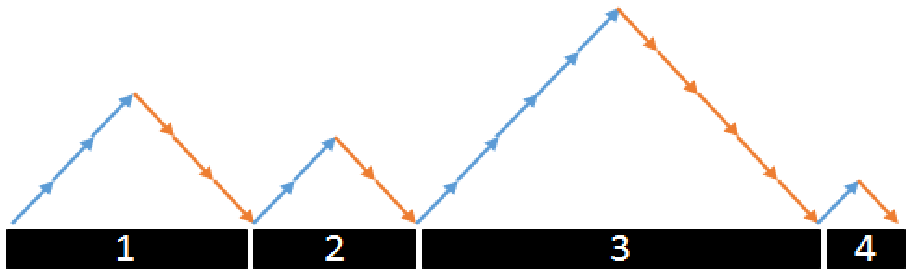

Figure 8: Three pyramid pairs $\{(1,2), (2,3), (3,4)\}$ in a flat Dyck path

Figure 8 shows three pyramid pairs in a flat Dyck path. The two left peaks in Figure 9 form a pyramid pair on level 2, but not level 1 or 0.

Consider an exact 0 cost path forming a pyramid pair. Intuitively, each of these pyramids are independent since **Digraph-flat-exact-paths** finds their exact 0 paths independently.

4.1 The general case

Consider a Dyck pyramid pair $\mathbf{a}^{k_1}\,\mathbf{a}^{-k_1}\,\mathbf{a}^{k_2}\,\mathbf{a}^{-k_2}$ for integers $k_1 \geq 1$, $k_2 \geq 1$. The word

$$\mathbf{a}^{s}\,\mathbf{a}^{k_1}\,\mathbf{a}^{-k_1}\,\mathbf{a}^{k_2}\,\mathbf{a}^{-k_2}\,\mathbf{a}^{-s},$$

has the *exterior pair* $\mathbf{a}^s$ and $\mathbf{a}^{-s}$, for an integer $s \geq 1$, see for example [41]. Exterior pairs are always made of pairs of matching elements. Semi-Dyck words have exterior pairs $\mathbf{a}^s$ and $\mathbf{a}^{-s}$ for any integer $s \neq 0$. Combining exterior pairs with flat grammars gives the general Dyck and semi-Dyck cases.

An *isolated* pyramid (valley) has no adjoining pyramid (valley). Pyramids and valleys are isolated by exterior pairs. If $\mathbf{a}^k\,\mathbf{a}^{-k}$ is an isolated pyramid, then it is enclosed by at least one exterior pair.

The rightmost pyramid in Figure 9 is an isolated pyramid. This isolated pyramid has label $\mathbf{a}^2\,\mathbf{a}^{-2}$. There is an isolated pyramid pair at base level 2. These two pyramids are on the left.

Definition 11 (Isolated paths) In a grid graph, an m *isolated path* is any maximal sequence of m pyramids or valleys all adjoining at the same base level.

A consequence of the definition of an m isolated path is it has no $(m+1)$st adjoining pyramid or valley. Isolated paths with $m = 2$ are *isolated pairs*. Isolated paths with $m = 4$ are *isolated quads*.

The four boxes in Figure 10 are isolated paths. There are also two pyramid pairs with three peaks in the middle. These pyramids are contained by an exterior pair.

An *invocation* of **Digraph-flat-exact-paths** runs $\lceil \log n \rceil$ iterations from line 4 in Figure 5. One invocation of **Digraph-flat-exact-paths** converts all of these isolated paths into exact 0 cost edges.

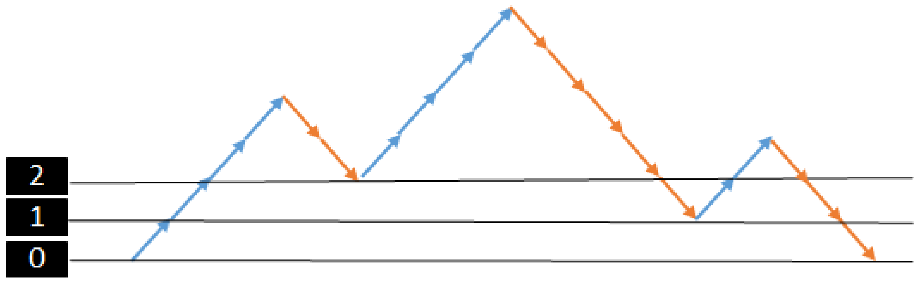

Figure 9: Level 2 has one isolated pyramid pair on the left, level 1 has an isolated pyramid on the right

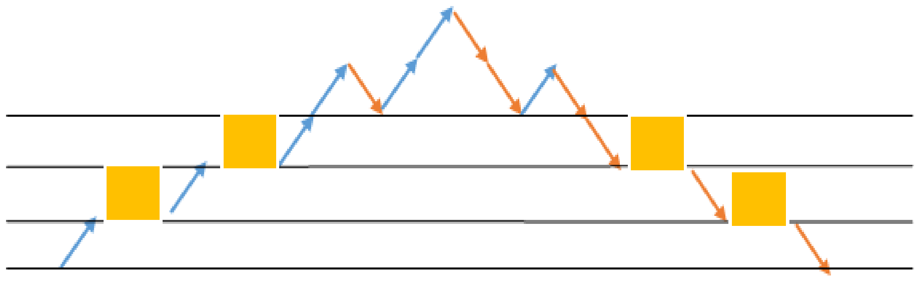

Figure 10: A Dyck or semi-Dyck path with four isolated paths in the boxes

Corollary 3 Given an LDG $G_1 = (\Sigma, V, E_1)$, where Σ is Dyck (semi-Dyck) and $|\Sigma| = 2$, then one invocation of **Digraph-flat-exact-paths** finds an exact 0 cost path for an m isolated path, where $m : n/2 \geq m \geq 2$.

Proof: Without loss, the focus is on an m isolated path of pyramids. Since the $m \leq n/2$ pyramids form an adjoining isolated path, they all have the same base level. This means the m pyramids form a flat Dyck path. So an invocation **Digraph-flat-exact-paths** computes the flat Dyck reachability of all adjoining pyramid pairs by Lemma 9. ∎

A Dyck *inner segment* is a Dyck word w between a set of exterior pairs $\mathbf{a}^s$ and $\mathbf{a}^{-s}$, for integers $s \geq 1$. The word $\mathbf{a}^s\, w\, \mathbf{a}^{-s}$ labels a valid Dyck path p and s is maximal so $\mathbf{a}^{s+1}\, w\, \mathbf{a}^{-s-1}$ does not label a valid path containing p. The semi-Dyck case has maximal exterior pairs $\mathbf{a}^s$ and $\mathbf{a}^{-s}$, for integers $s \neq 1$. A semi-Dyck inner segment is a semi-Dyck word.

Inner segments are contained by *pyramid (valley) bases* in the Dyck (semi-Dyck) case. If an inner segment is empty, then the exterior pairs $\mathbf{a}^s$ and $\mathbf{a}^{-s}$ ($\mathbf{a}^{-s}$ and $\mathbf{a}^s$) form a pyramid (valley), for $s \geq 1$.

Corollary 4 Given an LDG $G_\ell = (\Sigma, V, E_\ell)$, where Σ is Dyck (semi-Dyck) and $|\Sigma| = 2$ and $\ell \geq 2$, and suppose an 0 cost edge connects an inner segment of s exterior pairs. Then one invocation of **Digraph-flat-exact-paths** finds an exact 0 path through this inner segment and these exterior pairs.

Proof: Suppose a set of s exterior pairs contains a 0 cost edge. It must be that, $s < n/2$, and by assumption **Digraph-flat-exact-paths** already found inner segment's exact 0 cost path by iteration ℓ. Thus, **Digraph-flat-exact-paths** finds the exact 0 cost path including these exterior pairs by Lemma 9. ∎

Consider **Digraph-flat-exact-paths**. Finding exact 0 cost paths for isolated paths is not compatible with finding paths in their exterior pairs. This incompatibility is handled by careful iterations of **Digraph-flat-exact-paths**. See a single iteration in Figure 11. When needed, assume lines 3 and 4 are adapted for the Dyck case, see the discussion accompanying Figure 6.

Convention 2 (Atomic invocations of Digraph-flat-exact-paths) For the worst case of Figure 11, the algorithm **Digraph-flat-exact-paths** is atomic.

One invocation of **Digraph-flat-exact-paths**, Figure 5, does not find the exact 0 path from start to end of the top path in Figure 12. Similarly, one invocation of semi-Dyck version of **Digraph-flat-exact-paths** does not find the exact 0 cost path from start to end for the bottom path.

The next discussion illustrates the challenge of atomic invocations of **Digraph-flat-exact-paths**. Then we show how iterations of Figure 11 correct for these challenges.

```
1. $M \leftarrow$ **Digraph-flat-exact-paths**(G)  // Fig. 5 (semi-Dyck add Fig. 6)
2. $M \leftarrow M +$ **AdjMatrix**$(G)$  // original edges plus exact 0 paths
3. $M \leftarrow M^2$  // AGMY multiplication, extending $\pm 1$ edges with exact 0 paths
4. $M \leftarrow$ *Normalize_and_Divide_by_2*$(M)$
```

Figure 11: A single iteration for solving the general case, the general Dyck or semi-Dyck solution iterates these four steps $\lceil \log n \rceil$ times.

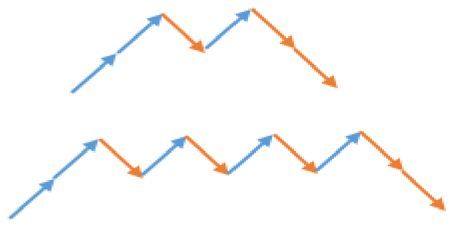

Figure 12: Cases where **Digraph-flat-exact-paths** alone does not work

Using label-costs and node names from 0 to 6, the top path in Figure 12 is,

$$p \quad = \quad 0 \xrightarrow{+1} 1 \xrightarrow{+1} 2 \xrightarrow{-1} 3 \xrightarrow{+1} 4 \xrightarrow{-1} 5 \xrightarrow{-1} 6.$$

The Dyck algorithm **Digraph-flat-exact-paths** fails to find the exact 0 path for p. To see this, let p_1 be the subpath made of the middle two edges, from node 2 to 4. So, p_1 is labeled with $\mathbf{a}^{-1}$ $\mathbf{a}$ and p_1 is not Dyck. In its first iteration, the algorithm finds ± 2 edges $0 \xrightarrow{+2} 2$ and $4 \xrightarrow{-2} 6$. Also the ± 2 edges are normalized to ± 1 edges: $0 \xrightarrow{+1} 2$ and $4 \xrightarrow{-1} 6$. In the Dyck case, there are no exact 0 paths to extend these new ± 1 edges, so they are removed in the second iteration. This leaves no exact 0 path from 0 to 6.

Consider the path p at the top of Figure 12. The semi-Dyck version of **Digraph-flat-exact-paths** finds the exact 0 cost path along p. This is because the middle two edges, from node 2 to 4, are labeled $\mathbf{a}^{-1}\,\mathbf{a}$. So in the first iteration, an exact 0 semi-Dyck path is found from 2 to 4. Also in this iteration, as in the Dyck case, a new $+1$ is created from 0 to 2. Likewise, a -1 edge is created from 4 to 6. All told, in line 9 of the first iteration, the new $+1$ edge is extended from node 0 to 4 and the -1 edge is extended to go from 2 to 6. Thus, the second iteration finds the semi-Dyck path from 0 to 6.

The flat semi-Dyck (Dyck) algorithm **Digraph-flat-exact-paths** does not find the exact 0 path for the bottom path of Figure 12. This is because its first iteration it does not find an exact 0 cost path from the first pyramid peak to the fourth pyramid peak. It does find the exact 0 path joining the three exact 0 cost valleys using line 9.

This same iteration extends the new ±1 edges after normalizing the ±2 edges at the start and end. The new $+1$ edge is extended to the second pyramid peak. The -1 edge is extended from the third pyramid peak. These extensions are all done by line 9. So they are computed in the same algebraic matrix multiplication that forms the exact 0 path joining the three valleys. Therefore, the extended ±1 edges cannot reach each other, so in the next iteration they cannot form an exact 0 cost path in line 5.

An iterative solution. The general solution is based on iterating invocations of **Digraph-flat-exact-paths**. See Figure 11. After each run of the flat path algorithm, all original ±1 edges are extended by the new exact 0 paths. Any new ±2 edges are normalized to ±1 edges in preparation for the next iteration. The process is repeated for finding more exact 0 paths. After each invocation of **Digraph-flat-exact-paths**, all new exact 0 paths remain for subsequent iterations.

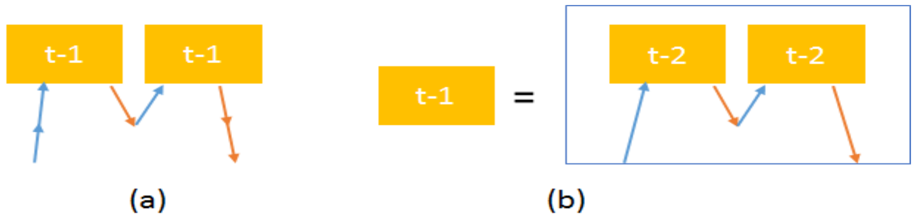

Figure 13: A worst case Dyck path for $t > 2$, where the $t = 1$ case is the empty block attaching its input edge directly to its output edge, giving $m = 2$ pyramids

In Figure 13(a), if the leftmost $t-1$ block is empty, the rightmost $t-1$ block continues recursively using the Figure 13(b), then the leftmost side is an isolated pyramid. In general, if the leftmost (rightmost) $t-1$ block is an isolated path, then it will be replaced by an exact 0 cost path in one invocation of **Digraph-flat-exact-paths** by Corollary 3. Each time this algorithm find an exact 0 cost path, a new 0 cost edges is created.

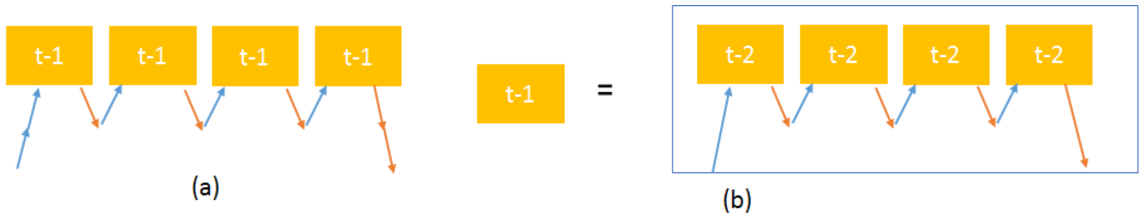

Figure 14: A worst case semi-Dyck path for $t > 2$, where the $t = 1$ case is the empty block attaching its input edge directly to its output edge giving $m = 4$ pyramids

This is because, for Dyck paths, Corollary 3 shows $m > 2$ pairs of adjoining pyramids require a single invocation of **Digraph-flat-exact-paths**. Just the same, a single invocation of **Digraph-flat-exact-paths** also finds an exact 0 path for pyramid pairs. Indeed, reducing $m > 2$ adjoining pyramids to $m = 2$ pyramids, frees vertices to contribute to worst case subpaths. Similarly, Corollary 4 indicates exterior pairs $\mathbf{a}^s$ and $\mathbf{a}^{-s}$, for $s \geq 1$, may be reduced to exterior pairs $\mathbf{a}$ and $\mathbf{a}^{-1}$ where $s = 1$. Since if $s > 1$ and say an exact 0 path is known for an inner segment, then one invocation of **Digraph-flat-exact-paths** finds the exact 0 path through these exterior pairs. Likewise, a single invocation finds the exact 0 path through exterior pairs $\mathbf{a}$ and $\mathbf{a}^{-1}$ when the exact 0 path is known for the inner segment. So, a reduced worst case Dyck path has all exterior pairs with $s = 1$.

Reduced semi-Dyck paths have exterior pairs $\mathbf{a}^s$ and $\mathbf{a}^{-s}$, for $s \in \{-1, +1\}$. Reduced valleys are labeled $\mathbf{a}^{-1}\,\mathbf{a}$. Finally, any $m > 4$ pairs of adjoining pyramids and/or valleys are replaced with $m = 4$ quads.

Worst case semi-Dyck paths are the exact 0 paths given in Figure 14. If there are only three pyramids in Figure 14(a), then they have two adjoining shared valleys. The first iteration line 5 creates a $+2$ edge at the start. Likewise, the first iteration creates a -2 edge at the end. These edges are normalized to new ± 1 edges in line 7. Finally, line 9 extends the new ± 1 edges using the two exact 0 paths just made from the two valleys. This gives a $+1$ edge adjoining a -1 edge for the next iteration. The next iteration of **Digraph-flat-exact-paths** combines these edges producing an exact 0 path from the start to the end of this path. In summary, semi-Dyck isolated paths with $m \leq 3$ pyramids and/or valleys become exact 0 paths in a single invocation of **Digraph-flat-exact-paths**.

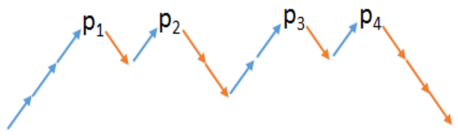

Figure 15: In the Dyck case, if the algorithm in Figure 11 converts the paths p_1, p_2, p_3 and p_4 into 0 cost edges, then these are two adjoining augmented pairs contained in an exterior pair

Definition 12 (Augmented pairs or quads) A Dyck (semi-Dyck) *augmented pair (quad)* is a path of two (four) adjoining pyramid (and/or valley) bases whose inner segments are 0 cost edges. All adjoining bases are the same base level and all adjoining bases are contained by an exterior pair.

If two augmented pairs adjoin at the same base level, then these augmented pairs become a single augmented pair in one iteration of Figure 11. Another iteration replaces this single augmented pair with a 0 cost edge. In general, say G' is a reduced Dyck grid graph with m augmented pairs all adjoining at the same base level. If all augmented pairs in G' have the same maximum level peaks, then all augmented pairs become exact 0 cost paths in the same iteration of Figure 11.

Figure 13 shows how augmented pairs may be constructed. In particular, each level of augmented pairs is contained by an exterior pair. A new augmented pair may only form if this augmented pair has an adjoint augmented pyramid and both together are contained by an exterior pair. The semi-Dyck augmented quads are similar, see Figure 14.

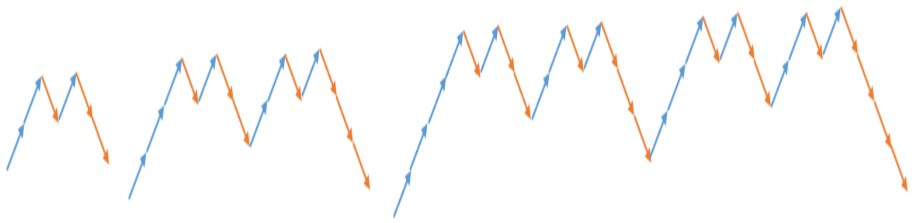
Figure 16: Three reduced worst case Dyck paths

Figure 16 gives reduced Dyck path examples with $m \in \{ 2, 4, 8 \}$ peaks. In general if $t(m)$ is the number of nodes n in a reduced Dyck path with m peaks, then $t(m) = 2\,t(m/2)+1$ with the base case $t(1) = 3$. This means $t(2) = 7$, $t(4) = 15$ and $t(8) = 31$. Or if $m = 2^k$, then $n = 2^{k+2} - 1$.

Lemma 10 Consider a reduced Dyck grid path with $m = 2^k$ pyramid peaks and $n = 2^{k+2} - 1$ nodes, for an integer $k \geq 1$. Suppose one pyramid peak is replaced by a 0 cost edge, then Figure 11 requires fewer than $\log m$ iterations to find the exact 0 paths.

Proof: In a grid graph let p be a reduced Dyck path with $m-1$ pyramid peaks, for $m = 2^k$ where $k \geq 1$. These peaks are all at maximum level m. Suppose, for the sake of a contradiction, determining that this path p is an exact 0 cost path requires $\log m$ iterations.

Since there are $m = 2^k - 1$ pyramid peaks at level m, there is at least one pyramid peak that is not in an isolated pair. This single isolated pyramid becomes an exact 0 cost path in the first iteration of Figure 11. So this exact 0 cost path cannot contribute to creating a new augmented pair by an invocation of **Digraph-flat-exact-paths**. Therefore, at the start of the second iteration of Figure 11, there is an augmented pair that does not have an equivalent neighboring pair to form a new augmented pair. Hence, in the third iteration, there is a new augmented pair just created from two augmented pairs that will not have an equivalent neighboring pair to form a new augmented pair.

In general, the after $\log m - 1$ iterations of this process, there are at least

$$m \sum_{i=1}^{\log m - 1} \frac{1}{2^i}$$

pyramid peaks that started at the maximum level m but they could not form a new augmented pair. This sum is larger than $\frac{1}{2}$ for $m \geq 4$ and for $m = 2$ peaks, dropping one gives a path with one peak which requires one invocation of the algorithm in Figure 11.

In summary, this means the remaining peaks already formed an exact 0 cost path. Thus, giving a contradiction since $\log m - 1$ iterations are sufficient to find this exact 0 cost path. ∎

In general suppose $t_s(m)$ is the number of nodes n in a reduced semi-Dyck path with $m = 4^k$ peaks, for $k \geq 0$. Therefore, $t_s(m) = 4t_s(m/4) - 1$ with the base case $t_s(1) = 3$. This means $t_s(4) = 11$ and $t_s(16) = 43$. Or if $m = 4^k$, then $n = \frac{2^{2k+1}+1}{3}$, see [42, A007583].

A proof of the next lemma is similar to the proof of Lemma 10.

Lemma 11 Consider a reduced semi-Dyck grid path with $m = 4^k$ pyramids or valleys and $n = \frac{2^{2k+1}+1}{3}$ nodes, for an integer $k \geq 1$. Suppose one pyramid or valley is replaced by an exact 0 path, then Figure 11 requires fewer than $\log_4 m$ iterations.

Lemmas 10 and 11 show the worst case Dyck or semi-Dyck paths have maximum peaks at height $O(\log m)$. Moreover, these results indicate it is sufficient to consider m peaks when m is a power of 2 for the Dyck case and powers of 4 for the semi-Dyck case.

Lemma 12 Given an LDG $G = (\Sigma, V, E_1)$, where Σ is Dyck (semi-Dyck) and $|\Sigma| = 2$, then a worst case path for iterations of Figure 11 is given by Figure 13 (Figure 14).

Proof: Without loss, consider only reduced Dyck paths.

All Dyck paths are built from pyramids. In reduced paths, all pyramids are in isolated pyramid pairs. The reduced pyramids are $\mathbf{a}\,\mathbf{a}^{-1}$. Furthermore, all isolated pyramid pairs have an exterior pair $\mathbf{a}^s$ and $\mathbf{a}^{-s}$ where $s = 1$.

By Lemma 10, the reduced worst case must start with $m = 2^k$ pyramid peaks, for $k \geq 1$, at maximum level m. The algorithm in Figure 11 finds the exact 0 path in such paths in $\lceil \log m \rceil \leq \lceil \log n \rceil$ iterations, since $m \leq n$.

A similar argument holds in the semi-Dyck case. The main difference is: semi-Dyck paths are built from pyramids and valleys. Moreover, adjoining isolated paths may be reduced to $m = 4$ adjoining isolated elements. Given these differences, all the Dyck arguments just presented remain the same.

This completes the proof. ∎

In the next lemma, height of a grid graph is the y value of maximum peak (x, y).

Lemma 13 Given an LDG $G = (\Sigma, V, E_1)$, where Σ is Dyck (semi-Dyck) and $|\Sigma| = 2$, then in the worst case finding all exact 0 paths takes $\lceil \log n \rceil$ iterations of Figure 11.

Proof: Without loss, the focus is on Dyck paths. Let $h(n)$ be the height of a worst-case Dyck path for iterations of the algorithm in Figure 11.

By Lemma 12, the worst-case paths must double their number of isolated pairs at each level. This means after the first iteration, each subsequent iteration of the algorithm in Figure 11 halves the number of augmented pairs.

Therefore, $h(n) \leq h(\lceil n/2 \rceil) + 1$ which immediately means $h(n) = O(\log n)$. In particular, the additive term of 1 is for each invocation of **Digraph-flat-exact-paths**.

▮

This section culminates in the main theorem.

Theorem 2 Given an LDG $G = (\Sigma, V, E_1)$, where Σ is Dyck (semi-Dyck) and $|\Sigma| = 2$, then Figure 11's algorithm solves Dyck (semi-Dyck) reachability in $O(n^{\omega} \log^3 n)$ time.

5 Determining ± 1 reachability

Determining ± 1 path reachability in an LDG is based on 0 cost edges computed by the algorithm in Figure 11. After running this algorithm, each 0 cost edge represents an exact 0 cost reachability path. This reachability is either Dyck and semi-Dyck reachability.

Definition 13 Let E^* contain all 0 cost edges found by running the algorithm in Figure 11 $\lceil \log n \rceil$ times.

The focus is on the E_1 edges of $G_1 = (\Sigma, V, E_1)$ in combination with the exact 0 cost edges E^*.

In the case of ± 1 reachability, consider the next paths from i to j. Both $i \longrightarrow k_1$ and $k_2 \longrightarrow j$ are edges in E_1 and $k_1 \overset{0}{\curvearrowright} k_2$ is an exact 0 cost edge from E^*. So, all cases for $i \xrightarrow{u} k_1$ and $k_2 \xrightarrow{v} j$ so that $u, v \in \{ -1, +1 \}$ are:

$$\begin{gathered}
i \xrightarrow{+1} k_1 \overset{0}{\curvearrowright} k_2 \xrightarrow{+1} j, \\
i \xrightarrow{+1} k_1 \overset{0}{\curvearrowright} k_2 \xrightarrow{-1} j, \\
i \xrightarrow{-1} k_1 \overset{0}{\curvearrowright} k_2 \xrightarrow{+1} j, \\
i \xrightarrow{-1} k_1 \overset{0}{\curvearrowright} k_2 \xrightarrow{-1} j.
\end{gathered}$$

If these are the only paths from i to j, then there is no ± 1 path from i to j.

Of course, all edges in E^* are built from edges in E_1. An edge is *directly* from E_1 if it is not in an exact 0 cost edge under discussion.

Lemma 14 Given an LDG $G_1 = (\Sigma, V, E_1)$, its exact 0 cost paths are 0 edges in E^* where Σ is Dyck (semi-Dyck) and $|\Sigma| = 2$, then all ± 1 paths can be found by extending all ± 1 edges in E_1 with only 0 cost edges in E^*.

Proof: Without loss, the semi-Dyck case is the focus. Recall, semi-Dyck paths are any paths with the same number of $+1$ and -1 edges from E_1.

Consider the edge $e_1 = i \xrightarrow{\pm 1} k_1 \in E_1$ and another edge $e_2 = k_2 \xrightarrow{u} j$ in E_1, where $u \in \{-1, +1\}$, so that there is a 0 cost edge from k_1 to k_2 in E^*. Let $k_1 \overset{0}{\curvearrowright} k_2$ be this 0 cost edge in E^*. Suppose, for the sake of a contradiction, that $i \xrightarrow{\pm 1} j$, is a ± 1 path that is not discovered by extending e_1 with edges from E^*. Consider the next cases.

Case 1: If e_1 and e_2 have different signs.

If e_1 and e_2 have different signs, then the entire path $i \xrightarrow{0} j$ is another 0 cost edge in E^* found by the algorithm in Figure 11. This is a contradiction, since in this case joining e_1 and e_2 does not create a ± 1 edge.

Case 2: If e_1 and e_2 have the same sign.

Given the exact 0 edge $k_1 \overset{0}{\curvearrowright} k_2$ from E^* between the edges e_1 to e_2 gives a ± 2 path: $p_{1,2} = \overset{\pm 1}{e_1} \overset{0}{\curvearrowright} \overset{\pm 1}{e_2}$.

Say the path $p_{1,2}$ contributes to a ± 1 edge in combination with another opposite sign edge e_3. There are two subcases that both lead to contraditions.

Subcase 2a: $\overset{\pm 1}{e_1} \overset{0}{\curvearrowright} \overset{\pm 1}{e_2} \overset{0}{\curvearrowright} \overset{\mp 1}{e_3}$.

This subcase gives $p_{2,3} = \overset{\pm 1}{e_2} \overset{0}{\curvearrowright} \overset{\mp 1}{e_3}$ which has exact cost 0. Thus, $p_{2,3}$ must be and edge in E^*. Therefore, the ± 1 reachabilty of $\overset{\pm 1}{e_1} \overset{0}{\curvearrowright} \overset{\pm 1}{e_2} \overset{0}{\curvearrowright} \overset{\mp 1}{e_3}$ requires only $e_1 = i \xrightarrow{\pm 1} k_1$ to be directly from E_1.

Subcase 2b: $\overset{\mp 1}{e_3} \overset{0}{\curvearrowright} \overset{\pm 1}{e_1} \overset{0}{\curvearrowright} \overset{\pm 1}{e_2}$.

This subcase gives $p_{3,1} = \overset{\mp 1}{e_3} \overset{0}{\curvearrowright} \overset{\pm 1}{e_1}$ which also has cost 0. Thus, $p_{3,1}$ must be in E^*. This means the ± 1 reachabilty of $\overset{\mp 1}{e_3} \overset{0}{\curvearrowright} \overset{\pm 1}{e_1} \overset{0}{\curvearrowright} \overset{\mp 1}{e_2}$ requires only $e_2 = k_2 \xrightarrow{\pm 1} j$ to be directly from E_1.

Since Dyck languages are also semi-Dyck, the proof is complete. ▮

Finding all exact 0 cost edges in an LDG G with all edges initially labeled from $\{ -1, 1 \}$ is the same as determining all 0 reachability in G.

Consider the output E^* from Figure 11. Lemma 14 indicates finding all exact ± 1 cost paths may be computed as the AGMY matrix product,

$$E^* \, M \, E^*$$

where M is the adjacency matrix of the given LDG $G = (V, E_1)$. This costs $\widetilde{O}(n^\omega)$.

6 Conclusion

Combining Theorem 2 with Lemma 14 gives useful results in finding exact Dyck and semi-Dyck paths in digraphs.

Starting with an LDG $G = (\Sigma, V, E_1)$, where Σ is Dyck with $|\Sigma| = 2$, then all $\{-1, 0, +1\}$ cost edges can be found in in $O(n^\omega \log^3 n)$ time. A number of powerful techniques can reduce this cost by polylog factors giving $\widetilde{O}(n^\omega)$, see [15, 38, 25, 18, 26]. In particular, Zwick [18] outlines such improvements nicely.

Acknowledgments

Thanks to Emily Proulx and Sarthak Behl for contributing example inputs and discussing applications of these results.

Thanks to Derek Morris for pointing out interesting applications as well as his encouragement.

References

[1] P. G. Bradford, "Efficient exact paths for Dyck and semi-Dyck labeled path reachability," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2017, to appear in the UEMCON proceedings, IEEE Press.

[2] M. Nykänen and E. Ukkonen, "The exact path length problem," *J. Algorithms*, vol. 42, no. 1, pp. 41–53, Jan. 2002.

[3] L. Lee, "Fast context-free grammar parsing requires fast Boolean matrix multiplication," *Journal of the ACM*, vol. 49, no. 1, pp. 1–15, 2002.

[4] C. Barrett, R. Jacob, and M. Marathe, "Formal language constrained path problems," *SIAM Journal on Computing*, vol. 30, no. 3, pp. 809–837, 2001.

[5] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.

[6] F. Afrati and C. Papadimitriou, "The parallel complexity of simple chain queries," in *Proc. of the 6th ACM Symposium on Principles of Database Systems (PODS '87).* New York, NY, USA: ACM, 1987, pp. 210–213.

[7] T. Reps, "On the sequential nature of interprocedural program-analysis problems," *Acta Inf.*, vol. 33, no. 5, pp. 739–757, Aug. 1996. [Online]. Available: http://dx.doi.org/10.1007/BF03036473

[8] J. D. Ullman and A. Van Gelder, "Parallel complexity of logical query programs," in *27th Annual Symposium on Foundations of Computer Science*, Oct 1986, pp. 438–454.

[9] M. Yannakakis, "Graph-theoretic methods in database theory," in *PODS '90: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems.* New York, NY, USA: ACM Press, 1990, pp. 230–242.

[10] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251 – 280, 1990.

[11] A. J. Stothers, "On the complexity of matrix multiplication," Ph.D. dissertation, University of Edinburgh, School of Mathematics, 2010.

[12] V. V. Williams, "Multiplying matrices faster than Coppersmith-Winograd," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 887–898.

[13] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '14. New York, NY, USA: ACM, 2014, pp. 296–303.

[14] D. Melski and T. Reps, "Interconvertibility of a class of set constraints and context-free-language reachability," *Theor. Comput. Sci.*, vol. 248, no. 1-2, pp. 29–98, Oct. 2000.

[15] N. Alon, Z. Galil, and O. Margalit, "On the exponent of the all pairs shortest path problem," *Journal of Computer and System Sciences*, vol. 54, no. 2, pp. 255 – 262, 1997.

[16] T. Takaoka, "Subcubic cost algorithms for the all pairs shortest path problem," *Algorithmica*, vol. 20, no. 3, pp. 309–318, 1998.

[17] Z. Galil and O. Margalit, "All pairs shortest paths for graphs with small integer length edges," *Journal of Computer and System Sciences*, vol. 54, no. 2, pp. 243 – 254, 1997.

[18] U. Zwick, "All pairs shortest paths using bridging sets and rectangular matrix multiplication," *J. ACM*, vol. 49, no. 3, pp. 289–317, May 2002.

[19] P. G. Bradford and D. A. Thomas, "Labeled shortest paths in digraphs with negative and positive edge weights," *RAIRO - Theoretical Informatics and Applications*, vol. 43, no. 3, pp. 567–583, 2009.

[20] P. G. Bradford, "Quickest path distances on context-free labeled graphs," in *6th WSEAS Int.Conf. on Info., Security and Privacy (ISP '07)*, 2007, pp. 22–29.

[21] P. G. Bradford and V. Choppella, "Fast point-to-point Dyck constrained shortest paths on a DAG (extended abstract)," in *2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, Oct 2016.

[22] E. Khamespanah, R. Khosravi, and M. Sirjani, "Efficient TCTL model checking algorithm for timed actors," in *Proceedings of the 4th International Workshop on Programming Based on Actors Agents & Decentralized Control*, ser. AGERE! '14. New York, NY, USA: ACM, 2014, pp. 55–66.

[23] C. B. Ward, N. M. Wiegand, and P. G. Bradford, "A distributed context-free language constrained shortest path algorithm," in *2008 37th International Conference on Parallel Processing*, Sept 2008, pp. 373–380.

[24] C. B. Ward and N. M. Wiegand, "Complexity results on labeled shortest path problems from wireless routing metrics," *Comput. Netw.*, vol. 54, no. 2, pp. 208–217, Feb. 2010.

[25] S. Chaudhuri, "Subcubic algorithms for recursive state machines," in *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '08. New York, NY, USA: ACM, 2008, pp. 159–169.

[26] W. Rytter, "Fast recognition of pushdown automaton and context-free languages," *Information and Control*, vol. 67, no. 1, pp. 12 – 22, 1985.

[27] H. Yuan and P. Eugster, *An Efficient Algorithm for Solving the Dyck-CFL Reachability Problem on Trees.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 175–189.

[28] Q. Zhang, M. R. Lyu, H. Yuan, and Z. Su, "Fast algorithms for dyck-cfl-reachability with applications to alias analysis," *SIGPLAN Not.*, vol. 48, no. 6, pp. 435–446, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2499370.2462159

[29] Q. Zhang and Z. Su, "Context-sensitive data-dependence analysis via linear conjunctive language reachability," *SIGPLAN Not.*, vol. 52, no. 1, pp. 344–358, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3093333.3009848

[30] N. Hollingum and B. Scholz, *Towards a Scalable Framework for Context-Free Language Reachability.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 193–211.

[31] A. Chakrabarti, G. Cormode, R. Kondapally, and A. McGregor, "Information cost tradeoffs for augmented index and streaming language recognition," *SIAM J. Comput.*, vol. 42, no. 1, pp. 61–83, 2013.

[32] H. Tang, D. Wang, Y. Xiong, L. Zhang, X. Wang, and L. Zhang, "Conditional dyck-cfl reachability analysis for complete and efficient library summarization," in *Programming Languages and Systems*, H. Yang, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 880–908.

[33] G. Grahne, A. Thomo, and W. W. Wadge, "Preferential regular path queries," *Fundam. Inform.*, vol. 89, no. 2-3, pp. 259–288, 2008.

[34] V. Choppella and C. T. Haynes, "Source-tracking unification." *Inf. Comput.*, vol. 201, no. 2, pp. 121–159, 2005.

[35] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3rd edition).* MIT Press, 2009.

[36] P. W. Purdom, "A transitive closure algorithm," *BIT Numerical Mathematics*, vol. 10, no. 1, pp. 76–94, 1970.

[37] G. Yuval, "An algorithm for finding all shortest paths using $n^{2.81}$ infinite-precision multiplications," *Inform. Process. Lett.*, vol. 4, no. 6, pp. 155–156, 1976.

[38] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1974.

[39] A. Shoshan and U. Zwick, "All pairs shortest paths in undirected graphs with integer weights," in *IEEE Foundations of Computer Science Conference.* IEEE, 1999, pp. 605–615.

[40] F. Romani, "Shortest-path problem is not harder than matrix multiplication," *Inform. Process. Lett.*, vol. 11, no. 3, pp. 134 – 136, 1980.

[41] A. Denise and R. Simion, "Two combinatorial statistics on dyck paths," *Discrete Mathematics*, vol. 137, no. 1, pp. 155 – 176, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0012365X93E0147V

[42] N. J. A. Sloane (editor). The on-line encyclopedia of integer sequences. Accessed: 2018-02-11. [Online]. Available: https://oeis.org