THESIS

FASTER GRAPH ALGORITHMS VIA SWITCHING CLASSES

Submitted by

Nathan Lindzey

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2012

Master's Committee:

Advisor: Ross McConnell

Wim Bohm
Tim Penttila

ABSTRACT

FASTER GRAPH ALGORITHMS VIA SWITCHING CLASSES

The runtime of an algorithm is intimately related to how an instance is represented. Recall that the runtimes of the first generation of graph algorithms were expressed as functions of $n := |V|$. This analysis was natural since at this time graphs were represented in n^2 space via their adjacency matrix. It was soon noticed that if $m := |E| = o(n^2)$, then a variety of graph algorithms could be sped-up by computing the adjacency-list from the adjacency matrix, then running the algorithm on the more efficient adjacency-list representation. This motivated the introduction of m to the runtime of graph algorithms and it is now customary in algorithm design to assume that a graph instance is given in the form of its adjacency-list. For instance, a graph algorithm is not considered to run in linear time unless it runs in $O(n + m)$ time. An $O(n^2)$ bound is not considered linear, even though the two bounds are the same in the worst case.

Let $\widetilde{m}$ be the size of the minimum representative of a graph G's switching class (w.r.t. to some switching operation). It is shown that better bounds for several classical graph algorithms can be obtained by modifying them so that their running time is a function of $n+\widetilde{m}$ rather than of $n+m$. This is significant because $\widetilde{m}$ is $O(m)$ but m is not $O(\widetilde{m})$. This is accomplished by first computing the so-called partially complemented adjacency list (pc-list) from an adjacency list, then designing an algorithm that is amenable to the more efficient pc-list representation. The pc-list data-structure is generalization of the adjacency list that has a natural correspondence to switching classes. Using this approach, better bounds are obtained for bipartite maximum matching, graph diameter, and vertex-weighted all-pairs shortest path.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

The runtime of an algorithm is intimately related to how an instance is represented. Consider the problem of determining the parity of an integer n. Before solving the problem, the representation of the instance n must be established. If n is given in unary representation, then determining the parity of n is linear in the number of ones. However, if n is given in binary representation, then the problem can be solved in constant time. Many problems that are not believed to be solvable in polynomial-time (e.g. integer factorization) can be solved in linear-time if the instance is represented in unary. This analysis is quite misleading since a number's unary representation is exponentially larger than it's binary representation. It is easy to see that there is no number theoretic problem for which a unary representation gives rise to a faster algorithm. In light of this, it is natural to claim binary to be a better representation and always insist that an instance is given in this form. This notion can be formalized as follows. Let X, X' be two representations for some class of objects C. If $|X'| = O(|X|)$ but $|X'| \neq \Theta(|X|)$, then there are members of C that have an asymptotically smaller representation under X', which makes X' a more efficient representation for that class. It is then possible for X' to be leveraged algorithmically to give rise to faster algorithms. The following brief historical overview of graph representations will help illustrate this point.

Runtimes of first generation graph algorithms were expressed as monomials of n. This analysis was natural since at this time graphs were represented via their $n \times n$ *adjacency matrix*. Of course, this representation is not an efficient way to represent digraphs since matrices take $\Theta(n^2)$ space and not all digraphs have size proportional to n^2.

It was soon noticed that if $m = o(n^2)$, then a variety of graph problems could be solved more efficiently by first computing an *adjacency list* representation from the adjacency matrix, then running an algorithm designed for the more efficient representation. This motivated the introduction of m to the runtime of graph algorithms and it is now customary in algorithm design to assume that a graph instance is given in the form of its adjacency-list.

The primary contribution of this work is the development of algorithms for the *partially complemented adjacency lists* (pc-lists) [3], a representation that is capable of realizing a variety of different *switching classes*. We show that several graph problems for which no linear-time algorithm is known can be seen as a switching class problem and solved more efficiently by first computing the pc-list from an adjacency list (or matrix) then running an algorithm designed for the pc-list. On the other hand, we give conditions for which it is fruitless to cast a graph problem as a switching class problem and end with future research directions.

1.1 Switching Classes

There are many types of graph switching operations in the literature [7] [3]. A property common to them all is that the operation alters the neighborhood of a set or single vertex by changing edges to nonedges and nonedges to edges.

Definition 1.1.1 *Let v be a vertex of directed or undirected graph G.*

- *An out-switch $\sigma_v^+(G)$ changes v's out-neighbors to non out-neighbors and vice versa.*
- *An in-switch $\sigma_v^-(G)$ changes v's in-neighbors to non in-neighbors and vice versa.*

Figure 1.2 shows the resultant *switched graph* after a sequence of out-switches on vertices $(2, 3, 4, 5, 6, 7, 8)$. Figure 1.2 shows the switched graph after the above sequence of out-switches and an in-switch on vertex 1. If we perform an in-switch and an out-switch on the same vertex then this is equivalent to what is known as Seidel-switching [7].

Definition 1.1.2 *A Seidel-switch $\sigma_v(G)$ changes v's neighbors to non neighbors and vice versa.*

Notice that if a Seidel-switch is performed on an undirected graph, then the switched graph is also undirected. Since the order in which we switch vertices does not matter, it is convenient to switch on a set $U \subseteq V$ rather a sequence which we denote by $\sigma_U(G)$. If both in-switches and out-switches are allowed, then two sets $I, O \subseteq V$ must be defined which we denote by $\sigma^{\pm}_{I,O}(G)$. This operation is known as *Gale-Berlekamp switching* for reasons which will soon be apparent. From these switching operations, there are a number of equivalence relations that one can define on $\mathcal{G}$, the class of labeled digraphs on n vertices.

Definition 1.1.3 *Let G, H be digraphs defined on the same labeled set of vertices V.*

- *G, H are out-equivalent if $\sigma^{+}_{U}(G) \cong H$ for some $U \subseteq V$.*
- *G, H are in-equivalent if $\sigma^{-}_{U}(G) \cong H$ for some $U \subseteq V$.*
- *G, H are in-out equivalent if $\sigma^{\pm}_{I,O}(G) \cong H$ for some $I, O \subseteq V$*
- *G, H are Seidel-equivalent if $\sigma_U(G) \cong H$ for some $U \subseteq V$.*

Proposition 1.1.4 *The relations above are equivalence relations over $\mathcal{G}$.*

Proof: Let $\sim$ denote the out-equivalent relation and let Δ denote symmetric difference. It's clear that $G \sim G$ since $\neg^{+}_{\{\}}(G) = G$. Since $\neg^{+}_{U}(\neg^{+}_{U}(G)) = G$, it follows that $G \sim H \Rightarrow H \sim G$. Lastly, if $H = \neg^{+}_{U}(G)$ and $F = \neg^{+}_{W}(H)$, then $\neg^{+}_{U \Delta W}(G) = F$. Proof that the rest of the relations form equivalences classes follows similarly. $\diamondsuit$

Let $\mathcal{G}^{+} := \mathcal{G}/\mathcal{C}^{+}$ be the out-switching classes on n vertices with members $G + \mathcal{C}^{+} \in \mathcal{G}^{+}$ and $H + \mathcal{C}^{+} \in \mathcal{G}^{+}$. Let $\bigoplus$ be the Hadamard product (symmetric difference) of the adjacency matrices of the coset representatives.

Theorem 1.1.5 $H = (\mathcal{G}^{+}, \bigoplus) \cong \mathbb{Z}_2^{n^2-2n}$

Proof: We have $|H| = 2^{n^2-2n}$ since each equivalence classes is of size 2^n and $|\mathcal{G}| = 2^{n^2-n}$. Since H is Abelian and $h^2 = 0 \ \forall h \in H$, it must be that $H \cong \mathbb{Z}_2^{n^2-2n}$. $\diamondsuit$

In fact, H is a vector space over $\mathbb{F}_2$. A basis for this vector space can be constructed as follows. Pick a graph B_v^1 for which the only adjacency is $v \to u$. Next, pick a graph B_v^2 for which the only adjacencies are $v \to u$, $v \to w$. Continue this process until v is connected to $n-2$ vertices. The neighborhood of v in any graph can be represented as a linear combination of elements of $\mathcal{B}$ and the vectors of $\mathcal{B}$ are linearly independent since they all lie in different equivalence classes. Repeating this process for all $v' \in V \backslash \{v\}$ will span H, so $\mathcal{B}$ is basis of size $n^2 - 2n$.

It is well-known that Seidel switching classes and Gale-Berlekamp switching classes afford groups [7] [16]. It is also clear that in-switching classes form a group since it is clear that $\mathcal{G}^- \cong \mathcal{G}^+$.

Definition 1.1.6 *The minimum representative of $\mathcal{C}$ is a not necessarily unique graph $\widetilde{G} \in \mathcal{C}$ having minimum edge cardinality $\widetilde{m}$.*

The minimum representative of an in-switching class $\widetilde{G}^- \in \mathcal{C}^-$ and an out-switching class $\widetilde{G}^+ \in \mathcal{C}^+$ can be found in $O(n+m)$. We can find $\widetilde{G}^+$ by out-switching every vertex whose neighborhood exceeds $n/2$ and $\widetilde{G}^-$ can be found by in-switching every vertex that appears more than $n/2$ times as an out-neighbor. It is then straightforward to perform the switches and construct $\widetilde{G}$ in $O(n^2)$ time.

Finding the minimum representative of an in-out equivalence class $\mathcal{C}^\pm$ corresponds to a well-known combinatorial and coding theory problem known as the *Gale-Berlekamp switching game* which is defined as follows. Let A be a $n \times n$ 0-1 matrix and let d be the total number of ones in A. The goal is to minimize d via switching (complementing) rows and/or columns of A [16]. It is NP-hard to compute $\widetilde{G}^\pm$; however, there do exist linear time approximation algorithms for this problem [11].

Seidel-switching classes (two-graphs) are a well studied object in the Algebraic Graph Theory that has a rich literature [7]. It has been shown that computing $\widetilde{G}$ of a Seidel-switching class is NP-hard [9], but can be approximated heuristically in $O(m)$ time as follows. Notice that if $u, v \in U$ are (non) adjacent in G, then u, v are (non) adjacent in $\sigma_U(G)$. It follows

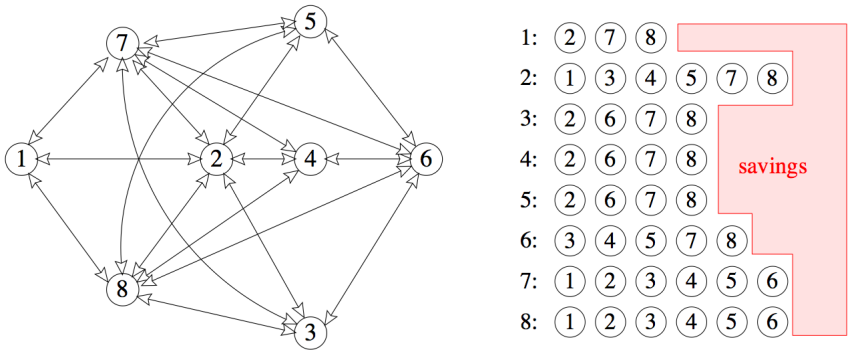

Figure 1.1: Undirected Graph G with Adjacency List

that $G[U] \cong \sigma_U(G)[U]$ and $G[V-U] \cong \sigma_U(G)[V-G]$ which implies that Seidel-switching on U is equivalent to complementing the cut induced by U. Therefore, given a dense graph G, one can find sparse members by approximating MAX-CUT which can be done in $O(m)$ time [14]. The problem can also be solved via a $(1-\epsilon)$ δ-dense approximation algorithm where δ is the smallest degree of any vertex in G [4]. This runtime of this algorithm can be $O(n^2)$ by setting ϵ accordingly.

1.2 Partially Complemented Adjacency Lists

A partially complemented adjacency list (pc-list) is an adjacency list outfitted with a boolean vector of size of n that represents vertex switches. If both in-switches and out-switches are permitted, then an additional boolean vector of size n is required. One of the virtues of this representation is that it preserves all the information of the original graph. Figures 1.1 1.2 1.3 demonstrate the space savings of this representation [3].

In [3] the question of whether symmetric (Seidel) switching classes could be leveraged algorithmically was left open. In the next chapter, we show that if the vertices are ordered such that the switched ones appear before unswitched ones and the neighbor-lists respect this ordering, then breadth-first search on a graph can be solved in time proportional to the minimal representative of its Seidel-switching class.

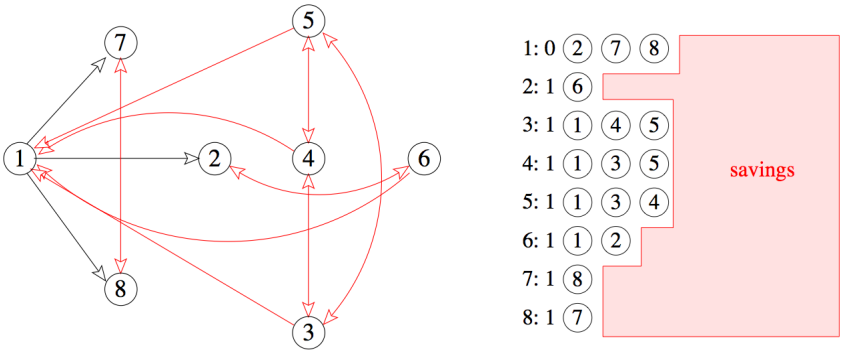

Figure 1.2: $\sigma_U^+(G)$, $U = \{2, 3, 4, 5, 6, 7, 8\}$ with pc-list

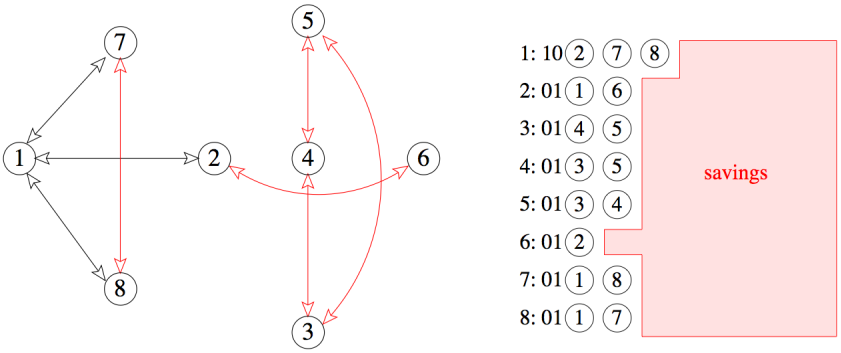

Figure 1.3: $\sigma_{I,O}^{\pm}(G)$, $I = \{1\}$, $O = \{2, 3, 4, 5, 6, 7, 8\}$ with pc-list

1.2.1 Limitations

To understand the limitations of partially complemented representations, it is useful to consider the Erdos-Renyi uniform random graph model $G(n, p = \frac{1}{2})$ [1]. In this model, an edge appears in a graph with probability 1/2 and so the probability of constructing an arbitrary labeled graph is $2^{-\binom{n}{2}}$. It is well-known that $G(n, p = \frac{1}{2})$ forms a binomial distribution over the class of graphs on n vertices and it follows that almost all graphs are $n/2$-regular by the law of large numbers. These graphs have approximately the same number of edges as nonedges so intuitively, one can see that switching them will not yield an asymptotically sparser representation. Since $|G| = \Theta(|\widetilde{G}|)$ almost always holds, we must concede that partially complemented representations do not provide an asymptotically better representation for most of graphs; however, this is not surprising since most graphs have a binary entropy that approaches one. What is more interesting is that in this setting the graphs which exhibit the worst runtimes are those that are $\approx n/2$-regular. This is contrary to traditional algorithm analysis which would have us believe that graphs denser than those that are $\approx n/2$-regular come closer to exhibiting worst-case behavior of an algorithm.

1.2.2 Advantages

Over the years there has been work towards speeding-up graph algorithms over dense instances. The techniques developed so far all somewhat involved insofar that they employ sophisticated pre-processing [6], data-structures [10], and "RAM tricks" [2] to achieve logarithmic improvements in runtimes for canonical graph problems over sufficiently dense graphs. It has also been shown that dense graphs can be approximated via sparse graphs with surprising accuracy. This is known as *graph sparsification* and many graph problems can be approximated in linear time using sparsification techniques which can provide huge (super-logarithmic) speed-ups in runtime [5]. The only drawback is that these techniques do not guarantee optimality. To our knowledge, this is the first graph compression or sparsification technique that is able to provide both super-logarithmic speed-ups in runtime and exact solutions for several canonical graph problems considered in [6] [10] [2] given a sufficiently

dense graph.

In our situation, a graph is sufficiently dense when $\widetilde{m} = o(m)$. To better understand the asymptotics, consider an arbitrary out-switching class $\mathcal{C}^+$ on n vertices. It is clear that $H \in \mathcal{C}^+ \Rightarrow \overline{H} \in \mathcal{C}^+$ so as a special case, when $\overline{H} = o(H)$, then it is possible for a pc-list to provide an asymptotic speed-up in runtime. We stress that if there exists a linear-time algorithm for solving the problem, then no asymptotic improvements can be made. For example, the results of [3] centered around the development of out-switching and in-switching class algorithms for graph problems that admit linear-time algorithms; however, it requires $O(n^2)$ time to construct $\widetilde{G}$ so there was no chance to make asymptotic improvements in runtime for those algorithms. It is important to note that although the speed-ups achieved [6] [2] are modest, there are dense graphs for which $\widetilde{m} = \Theta(m)$, so their methods can provide a logarithmic speed-up when ours cannot.

Finally, it is well known that solving many graph theoretical problems involves considering properties of both G and $\overline{G}$. A virtue of the pc-list is that it is a more natural representation for algorithmic problems where the complement must also be considered. In particular, the pc-list has been used to develop an elegant recognition algorithm for interval graphs. Interval graphs are exactly those graphs that are chordal and co-comparibility (graphs whose complements are transitively-orientable) [17]. In [12] a pc-algorithm for transitive orientation was developed which implied a $O(n + mlog(n))$ algorithm for computing a transitive orientation of a graph's complement. Since chordal graph recognition is $O(n + m)$ this gave rise to an elegant $O(n + mlog(n))$ interval graph recognition algorithm.

Chapter 2

Algorithms

There is an intuitive reason why pc-representations can provide asymptotically faster graph algorithms. Let's consider graph traversal on an undirected graph. The algorithm can be seen as a sequence of good and bad queries (edge visits). A good query discovers a vertex that has not been visited which results in adding a vertex and an edge to the traversal forest. A bad query discovers a vertex that has already been discovered which does not contribute to the output traversal forest. In the best case, an algorithm is composed entirely of good queries which results in a query sequence of length $n-1$ for connected graphs. Also, if the graph itself is a tree, then we are guaranteed this best case scenario. Let an optimal traversal for any graph be an $O(n)$ query sequence. Most traversal algorithms cannot guarantee an optimal traversal since it takes time to determine if a neighbor has already been visited. Let $\mathcal{O}$ be an oracle that provides an undiscovered non-neighbor of u or null for a current vertex u. By querying $\mathcal{O}$, we spend no time considering previously discovered vertices which gives an $O(n)$ algorithm. The problem is that without this oracle, there exists an ordering of the neighbor-lists where one must query all of the edges which makes traversal proportional to the number of edges in the worst case. Also, since there can only be $n-1$ good queries in any traversal algorithm, it follows that dense graphs have asymptotically more bad queries than good queries. The partially complemented representation allows us to get a smaller bound on the number of bad queries by charging them to $\widetilde{m}$ and guarantees that if a switched vertex queries a non-neighbor, then it will be a good query.

2.1 Basic Techniques and Algorithms

In this section we present basic techniques for designing pc-algorithms and review graph traversal for pc-lists presented in [3]. These techniques and traversal algorithms will be used to obtain better bounds and establish the correctness of more involved pc-algorithms. The bounds that we state are in terms of the size of the minimal representative $n + \widetilde{m}$; however, they can also be given in terms of the size of any member $\hat{G}$ of G's switching class. This is especially useful for switching classes where computing the minimal representative is computationally intractable.

- *The neighbor-lists can be radix-sorted with respect to an ordering of V in $O(n + \widetilde{m})$*
- *For any pc-algorithm, the number of non-neighbors that are examined must be proportional to $\widetilde{G}$.*

The later point is clear since if v is switched, then querying a non-neighbor u of v corresponds to querying the edge (v, u) in the original graph. If the number of queried non-neighbors is $\omega(\widetilde{G})$, then the runtime of the algorithm will be dominated by these queries.

In the algorithms below, the number of non-neighbors that are considered will be proportional to the number of vertices. To amortize the cost, each query to a non-neighbor will be charged to that non-neighbor and the sum non-neighbor charges will be $O(n)$ if each vertex is charged a constant number of times.

2.1.1 Breadth First Search

Theorem 2.1.1 *Given $\widetilde{G}$, it takes $O(n + \widetilde{m})$ to construct a BFS-tree of G.*

Assume that $\widetilde{G}$ is a minimal representative of G's out-switching class. As in the standard algorithm, divide the V into undiscovered nodes U, queued nodes, and processed nodes. Initially all nodes but the start node are in U, and the start node is inserted to a queue. To process a node, remove it from the front of the queue, and insert any undiscovered neighbors to the back of the queue, marking them as discovered. Keep the undiscovered nodes U in a

doubly-linked list. This list will allow us to amortize the cost of processing switched vertices. To process an non out-switched node, proceed as in the standard case. When processing an out-switched node v, mark the neighbors of v in U. Then traverse U, splicing out any unmarked nodes and inserting them on the queue. Marking the nodes can be charged to the edges of $\widetilde{G}$ that are responsible for applying the marks. Touching an unmarked node can happen at most once per node, since the node is removed from the list of undiscovered nodes whenever this happens. These two observations imply the $O(n + \widetilde{m})$ bound.

Now assume that $\widetilde{G}$ is a minimal representative of G's Seidel-switching class. Recall that for Seidel-switching we assume that the neighbor-lists of each vertex is ordered such that the switched vertices appear before the unswitched vertices. For convenience, we will add a dummy neighbor (marker) immediately before the first unswitched vertex of a neighbor-list. The BFS algorithm above is modified as follows. If the current vertex u is Seidel-switched, then every neighbor before the marker is handled as though u is not out-switched whereas every neighbor after the marker is handled as though v is out-switched. If the current vertex u is not Seidel-switched, then every neighbor before the marker is handled as though u is out-switched whereas every neighbor after the marker is handled as though u is not out-switched.

We now address a generalization of BFS known as the Single Source Shortest Path (SSSP) problem with respect to switching classes in Section 2.1.2.

2.1.2 Single Source Shortest Path

Briefly, optimization on switching classes is a somewhat obscure topic since optimization problems involving relations traditionally have been deemed graph theoretical. Most graph optimization problems involve computing minima or maxima over a weighted relation, hence they are usually not amenable to switching since the complement of a weighted edge is not well-defined. However, in some situations the weight of an edge (u, v) can be expressed as function of its endpoints (e.g. $w((u, v)) = w(u) + w(v)\ \forall u, v \in V$). This scenario is amenable to pc-representations because weights of nonedges are well-defined via their endpoints.

To determine if the weight function can be expressed as $w((u, v)) = w(u)+w(v)\ \forall u, v \in V$,

let $w : E \mapsto \mathbb{R}$ be a weight function and let I be the $m \times n$ edge-vertex incidence matrix of G. We can try to solve system of linear equations $I|\overrightarrow{w}$ where $\overrightarrow{w}$ are the edge weights in column vector form. When G is connected and contains more than n edges, then this corresponds to a sparse over-determined system of linear equations. In general, such a system admits no solutions so it follows that most edge weight functions cannot be represented as the sum of endpoints. However, if $I|\overrightarrow{w}$ does admit a solution, then single source shortest path problems can be solved more efficiently using pc-lists as follows.

Theorem 2.1.2 *Let G be a positive edge-weighted graph such that $w((u,v)) = w(u) + w(v)$. Given $\widetilde{G}^+$, `SSSP(G)` $= O(nlog(n) + \widetilde{m})$.*

Proof: Sort the vertices by weight, then radix sort the neighbor-lists with respect to the vertex ordering in time $O(nlog(n)+\widetilde{m})$. It is clear that if the doubly-linked list of undiscovered vertices U respects the ordering of V, then PC-BFS solves the problem in $O(n+\widetilde{m})$ time. It is also clear that if the vertex weights are positive integers $\leq N$, then `SSSP(G)` $= O(Nn+\widetilde{m})$

$\diamondsuit$

In other words, a shortest-path tree is a BFS-tree in this scenario which is generally not true for SSSP in general. This algorithm will be used in the next section to obtain a better bound for the All-Pairs Shortest Path (APSP) problem when the same restrictions are enforced on the weight function.

2.1.3 Depth-first Search

Theorem 2.1.3 *Given $\widetilde{G}$, it takes $O(n + \widetilde{m})$ to construct a DFS-tree of G.*

Proof: This bound was shown in [3]; however, the following DFS algorithm of [15] is conceptually much simpler. Assume that $\widetilde{G}$ is a minimal representative of G's out-switching class. As in the standard algorithm, divide V into undiscovered nodes U and processed nodes D. Initially all nodes but the start node s are in U and s is at the top of the stack D. Keeping U as a doubly-linked list will allow us to amortize the cost of processing complemented vertices. Assume the neighbors in the adjacency list in the same order as U and have each

neighbor n in a neighbor-list point to its corresponding node in U. To process an unswitched node v, proceed as in the standard case. To process a switched node, scan U from left to right skipping over neighbors of v until a non-neighbor u is encountered. Once u has been encountered, we remove u from U, push it onto D, then recurse. A non-neighbor can only be touched once since the node is immediately removed from U and pushed onto D. Skipping neighbors can be charged to the arc of G that is responsible. Notice that when u returns from a recursive call, the list of undiscovered vertices will be decimated by the recursive calls invoked by ancestors of u. This decimation makes it difficult to find the correct position in U for v to resume the scan for non-neighbors upon returning from a recursive call. To find a valid entrance into U, let n_v be v's most recently skipped neighbor before v invokes a recursive call. After v returns from the recursive call, if n_v is no longer in U, then n_v was pushed onto D by an ancestor of u. In this case, we remove n_v from N_v and set n_v to the next most recently skipped neighbor. We repeat this process until n_v points to a node that exists in U. If no such neighbor exists, then n_v points to the head of U. The node n_v now becomes the starting point of the scan in U. It is clear that every node to the right of n_v is either a non-neighbor of v or a neighbor that has not been touched. The time spent finding a valid entrance into U can be charged to the neighbor arcs of v. It is clear that each neighbor of v is skipped at most once since they are removed from N_v once they cannot be used to gain entrance to U. It follows that the total number of skips is $O(\widetilde{m})$ which gives a $O(n+\widetilde{m})$ time bound. $\diamondsuit$

2.2 Better Bounds for Super-Linear Graph Algorithms

This section is composed of new work that establishes better bounds for several super-linear graph algorithms.

2.2.1 Diameter

Definition 2.2.1 *The diameter of an unweighted graph is the maximum length of a shortest path between any two vertices.*

At present, the most efficient method for determining graph diameter is the $O(n(n+m))$ naive algorithm which calls BFS from each vertex and reports the height of the tallest BFS-tree across all runs. The theorem below shows that pc-representations easily yields a better bound for the problem.

Theorem 2.2.2 *The diameter of G can be computed in time $O(n(n+\widetilde{m}))$*

Proof: Construct $\widetilde{G}$ and run `PC-BFS`$(v, \widetilde{G})$ $\forall v \in V$ in $n^2 + n(n+\widetilde{m}) = O(n(n+\widetilde{m}))$. $\diamondsuit$

2.2.2 Vertex-Weighted All-Pairs Shortest Path

Theorem 2.2.3 *Let G be a positive edge-weighted graph such that $w_{uv} = w(u) + w(v)$. `APSP(G)` $= O(n(n+\widetilde{m}))$.*

Proof: Construct $\widetilde{G}^+$. Run `PC-SSSP` on an arbitrary vertex then run `PC-BFS` on each of the remaining vertices in time $n^2 + nlog(n) + n(n+m)) = O(n(n+\widetilde{m}))$ $\diamondsuit$

2.2.3 Bipartite Maximum Matching

At present, the best time bound for bipartite maximum matching is $O(\sqrt{n}m)$. We show that pc-representations allow for the problem to be solved in time $O(n^2 + \sqrt{n}\widetilde{m})$ or $O(\sqrt{n}\widetilde{m})$ if given $\widetilde{G}^+$. Since matching algorithms are among the most difficult combinatorial algorithms, we will assume out-switching for sake of simplicity. We begin with an overview of matching theory which can also be found in [8].

Definition 2.2.4 *A matching M is a set of edges such that no two edges are incident to the same vertex.*

Definition 2.2.5 *A path (without repeated vertices) is an augmenting path w.r.t. a matching M iff its endpoints are free (unmatched) and its edges are alternatively in E-M and in M.*

Lemma 2.2.6 *If M is a matching an P is an augmenting path with respect M, then $M\Delta P$ is a matching, and $|M\Delta P| = |M| + 1$.*

Proof: There is one more edge from $E - M$ than M in an augmenting path, hence those edges that don't belong to M become a new matching which is one greater than $|M|$. $\diamondsuit$

Theorem 2.2.7 *Let M and N be matchings s.t. $|M| = r$, $|N| = s$ and $s > r$, then $M\Delta N$ contains at least $s - r$ vertex disjoint augmenting paths w.r.t. M.*

Proof: Consider the subgraph H induced by the edges $M\Delta N$. Since no vertex of H has degree greater than two, H consists of components $C_1 \cdots C_k$ that are either isolated vertices, even cycles, or augmenting paths. Let $\delta(C_i)$ denote the number of N-edges minus the number of M-edges in C_i. If C_i is an isolated vertex or an even cycle, then $\delta(C_i) = 0$. If C_i is an augmenting path relative to M or N, then $\delta(C_i) = \{1, -1\}$. Since $\sum_i^k \delta(C_i) = s - r$, there must be at least $s - r$ augmenting paths relative to M that furthermore are vertex disjoint as they are each separate components. $\diamondsuit$

Theorem 2.2.8 *Let M and N be matchings s.t. $|M| = r$, $|N| = s$ and $s > r$, then $M\Delta N$ contains at least $s - r$ vertex disjoint augmenting paths w.r.t. M.*

Proof: Consider the subgraph H induced by the edges $M\Delta N$. Since no vertex of H has degree greater than two, H consists of components $C_1 \cdots C_k$ that are either isolated vertices, even cycles, or augmenting paths. Let $\delta(C_i)$ denote the number of N-edges minus the number of M-edges in C_i. If C_i is an isolated vertex or an even cycle, then $\delta(C_i) = 0$. If C_i is an augmenting path relative to M or N, then $\delta(C_i) = \{1, -1\}$. Since $\sum_i^k \delta(C_i) = s - r$, there must be at least $s - r$ augmenting paths w.r.t. M that furthermore are vertex-disjoint as they are each separate components. $\diamondsuit$

Corollary 2.2.9 *M is a maximum matching iff there is no augmenting path with respect to M.*

Corollary 2.2.10 *Let M be a matching, $|M| = r$, and $s > r$ be the cardinality of a maximum matching N. Then there exists an augmenting path with respect to M of length $\leq 2\lfloor r/(s-r)\rfloor + 1$.*

Proof: By Theorem 1, there are $s - r$ vertex-disjoint (hence edge-disjoint) augmenting paths w.r.t M. The augmenting paths together contain at most r M-edges, so any one of these augmenting paths has at most $\lfloor r/s - r \rfloor$ M-edges, hence at most $2\lfloor r/s - r \rfloor + 1$ altogether. $\diamondsuit$

Theorem 2.2.11 *Let M be a matching, P a shortest augmenting path w.r.t M, and P' an augmenting path w.r.t $M\Delta P$, then $|P'| \geq |P| + |P \cap P'|$.*

Proof: Let $N = M\Delta P\Delta P'$. Since $|N| - |M| = 2$, there are vertex-disjoint augmenting paths P_1, P_2 in $H = (V, M\Delta N)$. Since $M\Delta N = P\Delta P'$, we have $|P\Delta P'| \geq |P_1| + |P_2|$. But $P_1, P_2 \geq P$ since P_1, P_2 are disjoint and P is a shortest augmenting path w.r.t M. Hence, $|P\Delta P'| \geq |P_1| + |P_2| \geq 2|P|$ and since $P\Delta P' = |P| + |P'| - 2|P \cap P'|$, we have $|P'| \geq |P| + |P \cap P'|$. $\diamondsuit$

The result above suggests an iterative computing scheme for computing a maximum matching: find a shortest augmenting path P_i, set $M_i = M_i\Delta P_i$ and repeat, halting only when no augmenting path can be found. Another subtle corollary of Theorem 2 is the following:

Corollary 2.2.12 *For all i,j such that $|P_i| = |P_j|$, P_i and P_j are vertex-disjoint.*

Proof: Suppose $|P_i| = P_k$, $i < k$, P_i, P_j vertex-disjoint. Also, any P_j s.t $i < j < k$ is vertex-disjoint from P_i and P_k. This implies that P_k is an augmenting path w.r.t $M_i\Delta P_i$, and hence $P_k \geq |P_i| + |P_i \cap P_k|$ by Theorem 2. But $|P_i| = |P_k|$ implies $|P_i \cap P_k| = 0$, so P_i and P_j are edge-disjoint. Suppose P_i and P_k share a vertex v, but then the two paths must

share the $M_i \Delta P_i$-edge that is incident to v, a contradiction. $\diamondsuit$

From Theorem 2 and the above corollary, it follows that the lengths of each successive shortest augmenting path will be monotonic increasing and consecutive $P_i, P_{i+1}, \cdots P_j$ that aren't strictly increasing are all augmenting paths w.r.t. M_{i-1}.

Theorem 2.2.13 *Let s be the cardinality of a maximum matching. The number of distinct integers in the sequence, $|P_0|, |P_1|, \cdots, |P_i|, \cdots$ is less than or equal to $2\lfloor\sqrt{s}\rfloor + 2$.*

Proof: Let $r = s - \sqrt{s}$. Then $|M_r| = r$ and by Corollary 2, $|P_r| \leq 2\lfloor s - \sqrt{s}\rfloor/(s - \lfloor s - \sqrt{s}\rfloor) + 1 \leq 2\lfloor\sqrt{s}\rfloor + 1$. Keeping in mind the length of an augmenting path is always odd, there are at most $\lfloor\sqrt{s}\rfloor + 1$ distinct positive odd integers from $|P_0| \cdots |P_r|$. Also $|P_{r+1}| \cdots |P_s|$ contribute at most $\lceil\sqrt{s}\rceil$ distinct integers, so $\lfloor\sqrt{s}\rfloor + 1 + \lceil\sqrt{s}\rceil \leq 2\lfloor\sqrt{s}\rfloor + 2$. $\diamondsuit$

It follows that the number of distinct integers corresponds to the number of times Step 1 must be executed, so there are $O(\sqrt{n})$ executions of Step 1. Given the results above, the following is a general strategy for finding maximum matchings of arbitrary graphs.

MAX-MATCHING(G):

1. Let g be the length of a shortest augmenting path w.r.t M. Find a maximal set of vertex-disjoint paths $P_1 \cdots P_t$ each of length g that are augmenting paths w.r.t M. Otherwise, halt.
2. $M \leftarrow M\Delta P_1 \Delta \cdots \Delta P_t$; go to 1.

Finally, it suffices to show that Step 1 can be implemented in $O(n+m)$ to get a $O(\sqrt{n}(n+m))$ time-bound for computing a maximum matching. The following sections show how Step 1 can be implemented in $O(n + m)$ time for the bipartite case. We then show how the implementation can be modified to allow for partially complemented graphs which in turn will imply an $O(n + \widetilde{m})$ bound for Step 1.

Let $G^M = (A, B, E)$ denote the a bipartite graph with respect some matching $M \subseteq E$. The Hopcroft-Karp algorithm consists of $O(\sqrt{n})$ *phases* (calls to Step 1). At the beginning

of a phase, G is modified to contain a source vertex s that is adjacent to every unmatched vertex in A. A phase consists of a single call to BFS followed by a single call to DFS. For each of these calls we modify the traversal slightly as follows. If a current vertex of the search was reached via an unmatched edge, then the current vertex is forced to only visit its matched neighbor in the search. The call to BFS is necessary to build a directed acyclic level graph H having the property that any source to sink path is a shortest augmenting path. To ensure that the level graph has this property, BFS is not allowed to progress beyond the level where the first unmatched vertex in B is discovered. By Theorem [**?**], augmenting along a set of vertex-disjoint shortest augmenting paths at the end of each phase cannot decrease the depth of the level graph in the next phase. This explains why only $O(\sqrt{n})$ phases are needed. Once the level graph is determined, depth-first search is called on each neighbor of s to find a maximal set of vertex-disjoint shortest augmenting paths $\mathcal{P}$ in H. Once $\mathcal{P}$ has been computed, the matching M is extended by augmenting along each path in $\mathcal{P}$ and then a new phase begins. The algorithm halts when $\mathcal{P} = \emptyset$.

Theorem 2.2.14 *Let G be a bipartite graph. Then a maximum matching can be found in $O(n^2 + \sqrt{n}\widetilde{m})$ time.*

Proof:

There are a few obstacles that are immediately apparent for developing a partially complemented implementation of Hopcroft-Karp. Notice that in general if G is bipartite, then $\widetilde{G}$ is not bipartite, hence the graph implied by the pc-representation won't be bipartite. Furthermore,it is easy to see that no member of G's out-switching class can be asymptotically smaller than G. To overcome this, we can construct a *bipartite pc-representation* by restricting the set of non-neighbors of $v \in A$ to be $B - N(v)$ and maintaining two doubly-linked lists of undiscovered vertices U_A, U_B for the bipartition. It is clear that constructing H would require time disproportional to $\widetilde{m}$; however, this is not necessary since a level graph can be represented in $O(n)$ space by partitioning the vertices according to their level. Let L be an array of doubly-linked lists that represents said level partition.

Let F_A (F_B) be the set of unmatched vertices in A (B). We run PC-BFS from $s \in B$ with the following modifications. Let u be the current vertex at level i. If $u \in A$, then we use U_B for the list of undiscovered vertices. We splice out the newly discovered vertices as before, but we also insert the spliced vertices at $L[i+1]$. If u is matched and in B then we only visit the vertex that u is matched to in A. If a newly discovered vertex exists in F_B, then a shortest augmenting path P has been discovered. Once a shortest augmenting path has been discovered, we do not let PC-BFS progress past the current level and we only allow unmatched vertices to enter the queue. Once the search has been stopped, the queue contains only unmatched vertices of B reachable from free vertices of A via a shortest augmenting path. Once PC-BFS halts, the discovered vertices are partitioned as doubly-linked lists in L, but these lists do not necessarily respect the ordering of the neighbor-lists of vertices in A and B. To fix this, we can run a radix sort on ordered pairs (x, y) where x is the level and y is a vertex and reconstruct L in $O(n)$ time.

To find a maximal set of vertex-disjoint shortest augmenting paths, it suffices to call PC-DFS from each unmatched vertex in A, all of which reside in $L[1]$. It is easy to see that PC-DFS from a vertex at level i can be conducted per usual by setting $U = L[i+1]$ since each partition of L can be viewed as a decimated list. If the current vertex of PC-DFS is an unmatched vertex of B, then the DFS recursion stack corresponds to a shortest augmenting path P of H. In this case, we abort the current search, set $\mathcal{P} = \mathcal{P} \cup P$, remove the vertices of P from L, and then start a new DFS search from a free vertex in A. Once no more augmenting paths can be found, the matching M is updated in $O(n)$ by augmenting each $P \in \mathcal{P}$ which gives an $O(n + \widetilde{m})$ implementation of a phase. $\diamondsuit$

2.3 PC Amenability

In Section 2.1 we observed that optimization problems involving edge-weighted graphs in general cannot be solved more efficiently via pc-representation. We now explore other conditions for which pc-representations cannot be used to obtain more efficient algorithms. If the output

of a problem is $\Omega(m)$ space, then obviously any algorithm that solves the problem must take $\Omega(m)$ time. It follows that a problem is amenable to pc-representations if its output is $\omega(m)$. For this reason, problems such as Euclidean Tour and Ear Decomposition cannot be analyzed in terms of $\widetilde{m}$. Roughly speaking, if for every instance the problem requires every edge to be examined, then it cannot be solved more efficiently via pc-representations. However, the majority of graph problems have an $O(n)$ certificate so this is seldom an obstacle.

Even if the output of a problem is small enough, it is still not the case that any algorithm for the problem is amenable to pc-representations. It is possible for an algorithm to examine $\Theta(m)$ edges even though the output is $O(n)$. For instance, the maximum matching problem has an $O(n)$ output; however, the maximum matching algorithm of Micali and Vazirani for general graphs has insurmountable obstacles that make an $O(n+\widetilde{m})$ implementation of Step 1 unlikely [13].

Chapter 3

Conclusions

We have shown that when a super-linear graph problem can be cast as switching class problem, it is possible for the pc-list to provide a better bound for algorithm that solves the problem. Aside from obtaining better bounds, pc-lists also benefit algorithmic graph theory by allowing $O(n^2)$ factors to be removed from runtimes that arise when one has to construct the complement to solve a problem posed on the original graph. This plays an important role in graph class decision problems since a linear recognition algorithm for $\mathcal{C}$ does not imply a linear recognition algorithm for co-$\mathcal{C}$ since one must first construct the complement. However, if the algorithm is amenable to pc-representations, then it recognizes $\mathcal{C}$ and its complement in the same time and space.

3.1 Future Work

A pc-algorithm for general graph maximum matching has been developed, but proof of its correctness has been put aside as doctoral work. We previously mentioned that the algorithm of Micali and Vazirani is unlikely to be amenable to pc-representations, hence our pc-algorithm is modification of Gabow and Tarjan's $O(\sqrt{n}m)$ algorithm. Their algorithm is remarkable since it requires $O(\sqrt{n})$ phases but unlike the algorithms of Micali-Vazirani and Hopcroft-Karp, it does *not* require that the vertex-disjoint augmenting paths of Step 1 be shortest. A pc-algorithm for general graph maximum matching immediately allows for a better bound on the *k-edge connectivity augmentation problem* when $k = n - 2$. The k-edge connectivity augmentation problem asks for smallest set of edges F such that $G' = (V, E \cup F)$

is k-edge connected. This problem is NP-hard; however for the special case of $(n-2)$-edge connectivity, the problem reduces to finding a maximum matching on the complement which makes it solvable in $O(n^2+\sqrt{n}\overline{m})$ [18]. A pc-algorithm for general maximum matching would solve this problem in $O(\sqrt{n}m)$ time or $O(n^2+\sqrt{n}\widetilde{m})$.

We conclude with an agenda for future doctoral work on the subject of pc-representations:

- Establish correctness of partially complemented implementation of Gabow-Tarjan.
- Determine if negative weights can be handled in PC-SSSP and PC-APSP.
- Develop vertex-weighted bipartite and general maximum matching for switching classes.
- Further development of algorithms for Seidel-switching classes.

REFERENCES

[1] Bela Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[2] Joseph Cheriyan and Kurt Mehlhorn. Algorithms for dense graphs and networks on the random access computer. *Algorithmica*, 15(6):521–549, 1996.

[3] Elias Dahlhaus, Jens Gustedt, and Ross M. McConnell. Partially complemented representations of digraphs, 1999.

[4] W. Fernandez de la Vega. Max-cut has a randomized approximation scheme in dense graphs. *Random Struct. Algorithms*, 8(3):187–198, May 1996.

[5] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, September 1997.

[6] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51(2):261–272, October 1995.

[7] Chris Godsil and Gordon Royle. *Algebraic Graph Theory*. Springer, April 2001.

[8] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[9] Eva Jelínková, Ondrej Suchý, Petr Hlinený, and Jan Kratochvíl. Parameterized problems related to seidel's switching. *Discrete Mathematics & Theoretical Computer Science*, 13(2):19–44, 2011.

[10] Ming-Yang Kao, Neill Occhiogrosso, and Shang-Hua Teng. Simple and efficient graph compression schemes for dense and complement graphs. *J. Comb. Optim.*, 2(4):351–359, 1998.

[11] Marek Karpinski and Warren Schudy. Linear time approximation schemes for the galeberlekamp game and related minimization problems. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 313–322, New York, NY, USA, 2009. ACM.

[12] Ross M. Mcconnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Math*, 201:189–241, 1995.

[13] Silvio Micali and Vijay V. Vazirani. An $o(\sqrt{v}e)$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980.

[14] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[15] Nissa Osheim. *Personal Communication*.

[16] R. M. Roth and K. Viswanathan. On the hardness of decoding the gale berlekamp code. *IEEE Trans. Inf. Theor.*, 54(3):1050–1060, March 2008.

[17] Jeremy Spinrad. *Efficient Graph Representations*. AMS, 2003.

[18] Laszlo Vegh. *Connectivity Augmentation Algorithms*. PhD thesis, Eotvos Lorand University, 2010.