UNARY MINIMUM COST PATH PROBLEMS, ALTERNATING LOGSPACE, AND RUZZO, SIMON, AND TOMPA'S DL^{NL}

Rodney R. Howell, Louis E. Rosier,
and Hsu-Chun Yen

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-13 May 1987

Unary Minimum Cost Path Problems, Alternating Logspace, and Ruzzo, Simon, and Tompa's $\mathrm{DL}^{\mathrm{NL}}$

Rodney R. Howell, Louis E. Rosier, and Hsu-Chun Yen

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

In this paper, we examine the class $\mathrm{DL}^{\mathrm{NL[log]}}$ -- the class of languages accepted by O(log n) space bounded deterministic oracle Turing machines that are allowed to make at most O(log n) calls to an NL oracle. We show that this class is equivalent to the alternating logspace hierarchy, which has recently been shown by Lange, Jenner, and Kirsig to collapse to its second level (i.e., $\mathrm{A}\Sigma_2^{\mathrm{L}}$). Our results provide a succinct proof of this collapse as well as a deterministic characterization of $\mathrm{A}\Sigma_2^{\mathrm{L}}$. This characterization readily allows us to show a number of "natural" problems related to the minimum cost path in a directed graph with unary edge weights to be complete for $\mathrm{A}\Sigma_2^{\mathrm{L}}$. As we examine these problems, we show variations to be complete for NL and D^{L} -- the class of languages that can be expressed as the intersection of a language in NL with a language in co-NL. In the process, we point out the characteristics of the problems that appear to trigger the respective jumps in complexity. Using techniques introduced by Krentel, we then give a characterization of $\mathrm{DL}^{\mathrm{NL[log]}}$ in terms of optimization problems. Since problems that have been shown in the past to be complete for $\mathrm{A}\Sigma_2^{\mathrm{L}}$ do not appear to be optimization problems, but instead have to do with finding particular types of strongly connected components, our work unifies two groups of problems which appear, at least on the surface, to be very different. Finally, we examine how the techniques of Lange, Jenner, and Kirsig may be applied to other related hierarchies.

1. Introduction

The logspace oracle hierarchy and the logspace alternation hierarchy were introduced in Ruzzo, Simon, and Tompa [23], and Chandra, Kozen, and Stockmeyer [3], respectively. Both of these hierarchies are contained in PTIME[1]$\cap$DSPACE($\log^2$ n). Let $\mathrm{DL}^{\mathrm{NL}}$ denote the class of languages accepted by O(log n) space bounded deterministic *oracle Turing machines* (hereafter, OTMs), of the type prescribed in [23], whose oracles are from NL -- i.e., Δ_2^{L} in the logspace oracle hierarchy. Ruzzo, Simon, and Tompa showed in [23] that the entire logspace alternation hierarchy is contained within $\mathrm{DL}^{\mathrm{NL}}$. More recently, Lange, Jenner, and Kirsig [15] showed that $\mathrm{A}\Sigma_2^{\mathrm{L}}$ coincides with the closure of NL under Hausdorff reductions -- denoted by $\mathcal{L}_{\mathrm{hd}}$(NL) in [15]. Since the latter is closed under complementation, [15] yields the remarkable result that the alternating logspace hierarchy collapses.

Let $\mathrm{DL}^{\mathrm{NL[f]}}$ denote the class of languages accepted by O(log n) space bounded deterministic OTMs

[1]For brevity, hereafter, we use the abbreviations DL, NL, and P to denote the complexity classes DLOGSPACE, NLOGSPACE, and PTIME, respectively.

whose oracles are from NL, where all computations are restricted to allow at most O(f(n)) oracle calls. In this paper, we examine closely the class $DL^{NL[\log]}$. We first provide, in Section 3, a simple proof that $DL^{NL[\log]}=A\Sigma_2^L$. Since $DL^{NL[\log]}$ is clearly closed under complementation, our strategy yields a short proof that the alternating logspace hierarchy collapses. (Tompa [25] has also synthesized a short proof of the result in [15].) More importantly, it provides a deterministic computational model characterizing $A\Sigma_2^L$, $\mathcal{L}_{hd}(NL)$, and the alternating logspace hierarchy. The characterization is especially useful when considering problems related to the hierarchy, as it allows one to provide the analysis using another computational model. We subsequently illustrate the characterization's usefulness, in Section 4, when exploring the complexity of a number of "natural" problems -- many of which turn out to be complete for $DL^{NL[\log]}$.

One claim that we substantiate in this paper is that $A\Sigma_2^L$ is a robust complexity class. One measure of robustness revolves around whether a complexity class can be shown to contain many "natural" complete problems. Clearly, NL, P, NP, and PSPACE are robust, in this sense. More recently defined complexity classes that fall into this category might include D^P, $P^{NP[\log]}$, P^{NP}, and NP^{NP}. (D^P is the class of languages each of which can be expressed as the intersection of a language in NP with a language in co-NP. $P^{NP[\log]}$ is the class of languages accepted by deterministic polynomial time bounded OTMs, with oracles from NP, where all computations are restricted to allow at most O(log n) oracle calls. P^{NP} and NP^{NP} are Δ_2^P and Σ_2^P, respectively, in the polynomial time hierarchy [24].) For examples of natural complete problems for these classes the reader is encouraged to consult Bentley, Ottman, and Widmayer [2], Howell and Rosier [8], Huynh [9], Krentel [11], Papadimitriou [17], and Papadimitriou and Yannakakis [18]. Krentel's paper is especially interesting as it provides a characterization of $P^{NP[\log]}$ and P^{NP} in terms of functions whose ranges are not restricted to {0,1}. In particular, Krentel shows that "the inherent complexity of UNIQUE OPTIMAL TRAVELLING SALESPERSON, as considered by Papadimitriou [17], really comes from computing the length of the optimal tour. The uniqueness serves only to transform the problem into a decision problem."

We propose that the class $A\Sigma_2^L$ (or $DL^{NL[\log]}$) should also be considered robust (in the aforementioned sense). Natural complete problems for this class have been illustrated in Howell, Rosier, Huynh, and Yen [7], Lange [13], and Rosier and Yen [21, 22]. To further substantiate the above claim, we consider, in Section 4, the complexity of a number of problems involving directed graphs with *unary* edge weights. We give completeness results for several problems related to computing the *minimum cost path* in such a graph. Variations are later considered for problems concerning minimum *length* paths (i.e., the case where all edge weights are of cost 1), as well as problems involving the eccentricity of a vertex, the radius of a vertex, and the center of a graph. (See Aho, Hopcroft, and Ullman [1] for a precise definition of these terms.) One of the reasons for our interest in these graph problems stems from the fact that they seem very central to the study of computer science. For example, Aho, Hopcroft, and Ullman [1] devote an entire chapter of their textbook on data structures to the discussion of directed graph problems. Furthermore, in [4], Cook mentions parallel algorithms for several of these problems where unary edge weights are assumed. An example of a problem we show to be $DL^{NL[log]}$-complete is that of determining the parity of the minimum cost path from u to v in a graph G. As a second example, given a graph G and a vertex v the problem of determining whether v is a "center" of G is also $DL^{NL[log]}$-complete.

In the process of providing our complexity results, we examine a sequence of related problems that are complete for NL, D^L, or $DL^{NL[log]}$. (D^L is analogous to D^P, and is defined as the class of languages each of which can be expressed as the intersection of a language in NL with a language in co-NL.) Some of our complexity results are somewhat surprising; for example, the problem of verifying a potential solution to a single source, single sink minimum cost path problem is D^L-complete, whereas the problem of verifying a potential solution to an all pairs minimum cost path problem is NL-complete, and is in DL if all edge weights are restricted to be 1. As we proceed, we examine just what seems to trigger each jump in complexity, keeping in mind what seems to cause the analogous jumps in complexity from NP to D^P to P^{NP}. Many of the problems we consider seem particularly well suited for $DL^{NL[log]}$. (As a result, we showed some of them to be complete for $DL^{NL[log]}$ long before we realized that $DL^{NL[log]}=A\Sigma_2^L$.) Also, the inherent complexity of such problems seems to come from the necessity of computing optimal cost paths -- in much the same way as the inherent complexity of $P^{NP[log]}$ (P^{NP}) complete problems was due to the necessity of computing the value of a certain optimization function (see [11]). Hence, the class $DL^{NL[log]}$ seems to bear a great deal of similarity to $P^{NP[log]}$ and P^{NP}. Some of the proofs are also interesting in that they are simplified by the $DL^{NL[log]}=A\Sigma_2^L$ characterization. This may be of special interest since the previously studied $A\Sigma_2^L$-complete problems [7, 13, 21, 22] do not appear to be optimization problems. (Most of these problems involve finding particular types of strongly connected components in directed graphs.) Thus, the equivalence of $DL^{NL[log]}$ to $A\Sigma_2^L$ unifies two groups of problems which, on the surface, appear to be different.

Finally, in Section 5, we take a look at whether the techniques utilized here (and in Lange, Jenner, and Kirsig [15]) can be adapted to yield the collapse of similarly defined space hierarchies. As a result, other variations of the basic strategy are developed in order to illustrate that similarly defined alternating hierarchies collapse for other space classes -- e.g., $O(\log^2 n)$ and $O(n)$. We then turn our attention to the problem of whether the related symmetric complementing logspace hierarchy of Reif [20] collapses. Although we are unable to answer this question, we do establish that $DL^{SL[log]}=\mathcal{L}_{hd}(SL)$ -- the closure of SL under Hausdorff reductions. (SL denotes symmetric logspace. See Lewis and Papadimitriou [16] and Reif [20]. $DL^{SL[log]}$ and $\mathcal{L}_{hd}(SL)$ are defined analogously to $DL^{NL[log]}$ and $\mathcal{L}_{hd}(NL)$.) Hence, with respect to this hierarchy, the strategies employed here are once again essentially equivalent to those in [15]. Whether the strategy will ultimately work seems to depend on whether certain sets that involve counting (with respect to undirected graphs) can be recognized in SL. In particular, the relevant proofs in this paper and in [15, 25], revolve around being able to count reachable states in a graph. This counting (with respect to undirected graphs) does not seem possible in SL.

In what follows, we assume the reader is familiar with the basic tenets of automata and complexity theory. Relevent sources might include [5, 6]. The basic computational model used in this paper is the *nondeterministic (deterministic) offline multitape Turing machine.* All completeness classes mentioned in this paper are with respect to deterministic logspace many-one reductions.

The remainder of this paper is organized as follows. In Section 2, we provide most of the formal definitions used in the paper and illustrate many of the known relationships between the various complexity classes. In Section 3, we show that $DL^{NL[\log]}=A\Sigma_2^L$. Section 4 concerns itself with the complexity of the various graph problems. Special attention here is paid to the analogies with the work of Krentel [11]. Section 5 concludes by exploring whether the techniques utilized thus far can be used to induce the collapse of other "space" hierarchies.

2. Definitions of the relevant complexity classes

Ruzzo, Simon, and Tompa introduced the logspace oracle hierarchy in [23]. Informally, an *oracle Turing machine* (OTM) M is a nondeterministic (or deterministic) offline multitape Turing machine with an additional write-only tape called a *query tape* and three distinguished states called *query, yes,* and *no states.* In addition, a set A, called the *oracle set* is always related to the computation of the OTM M. Let $L(M^A)$ denote the language accepted by the OTM M using A as its oracle set (denoted M^A). The computation of an OTM is similar to that of an ordinary NTM except when visiting the query states. When in a query state, if the string on the query tape is in the oracle set, then the machine enters the yes state; otherwise, it enters the no state. Moreover, the contents of the query tape will be erased immediately upon entering the yes or no state. The machine can write a symbol on the query tape in every state except the query state. Now let X and Y be complexity classes. We define X^Y as the set of languages that can be recognized by an OTM M with oracle set A such that M operates within the complexity constraints of X, and A is in Y. Also, if f is any function on the natural numbers, $X^{Y[f]}$ is defined as the set of languages that can be recognized by an OTM M with oracle set A such that M operates within the complexity constraints of X, A is in Y, and X makes at most O(f(n)) oracle calls to A.

In order to study complexity classes between NL and P, it seems natural to define classes DL^{NL}, NL^{NL}, etc., thereby emulating the strategy of Stockmeyer [24] in defining the polynomial time hierarchy between NP and PSPACE. Unfortunately, when the OTM's are defined as above, one can easily show that $NL^{NL}=NL^{DL}=NP$. That is, the logspace hierarchy jumps into the polynomial time hierarchy on the second level. As is pointed out in [12, 23], this phenomenon occurs because nondeterminism when combined with the ability to write long strings on the query tape boosts the machine's computing power dramatically. To limit the undesired boost in the power of the OTM's, Ruzzo, Simon, and Tompa [23] require that only strings of logarithmic length be written on the query tape. However, the action of the query states is modified so that if machine M enters its query state with x on its input tape and y on its query tape, then M enters its yes state iff x#y (rather than y) is in the oracle set. (This definition actually utilizes the alternate characterization provided by Lemma 7 in [23].) **When considering sublinear space complexity classes, we consider only OTM's of this sort.** Using such OTM's, the logspace oracle hierarchy was defined in [23] as follows:

- $\Sigma_0^L = \mathrm{DL}$
- $\Sigma_{k+1}^L = \mathrm{NL}^{\Sigma_k^L}$
- $\Delta_{k+1}^L = \mathrm{DL}^{\Sigma_k^L}$
- $\Pi_k^L = \text{co-}\Sigma_k^L$

In [3], the alternating logspace hierarchy was introduced based on the model of *alternating Turing machines* (ATM's). Basically the concept of alternation is a generalization of nondeterminism in a way that allows existential and universal quantifiers to alternate during the course of a computation. Four kinds of states exist in an ATM; namely existential, universal, accepting, and rejecting states. A universal state can lead to acceptance iff all successors lead to acceptance. On the other hand, an existential state leads to acceptance iff there exists a successor that leads to acceptance. Details can be found in [3]. In particular, $A\Sigma_f^L$ is the set of languages accepted by O(log n) space bounded ATM's in which the starting state is an existential state and the machine is constrained to make at most f(n)-1 alternations during the course of a computation. (Any computation path of such a machine will have at most f(n) alternation blocks.) In [3], such machines are called "f(n) alternation bounded." We adopt this notation here.

The alternating logspace hierarchy was then defined to be $\cup_{k\geq 1} A\Sigma_k^L$. (Here the subscript k denotes the constant function f(n)=k.) The results of [15] show that $\mathcal{L}_{hd}(\mathrm{NL}) = A\Sigma_2^L = \cup_{k\geq 2} A\Sigma_k^L$, where $\mathcal{L}_{hd}(\mathrm{NL})$ is the closure of NL under Hausdorff reductions, which we will now define. Let A and B be two languages. A is said to be *Hausdorff reducible* to B iff there exists an O(log n) space bounded deterministic transducer M that produces, on input x, a list of strings $y_1, y_2, \ldots, y_{2*n_x}$ such that

1. $x \in A$ iff $y_{2*i-1} \in B$ and $y_{2*i} \notin B$ for some i, $1 \leq i \leq n_x$, and

2. for all i, if $y_{i+1} \in B$, then $y_i \in B$.

$\mathcal{L}_{hd}(\mathrm{NL})$ is now defined as the closure of NL under Hausdorff reducibility.

Combining together the results of many sources (including this paper), we now provide a sequence of inclusions that will help the reader to put many of these complexity classes into a clear perspective.

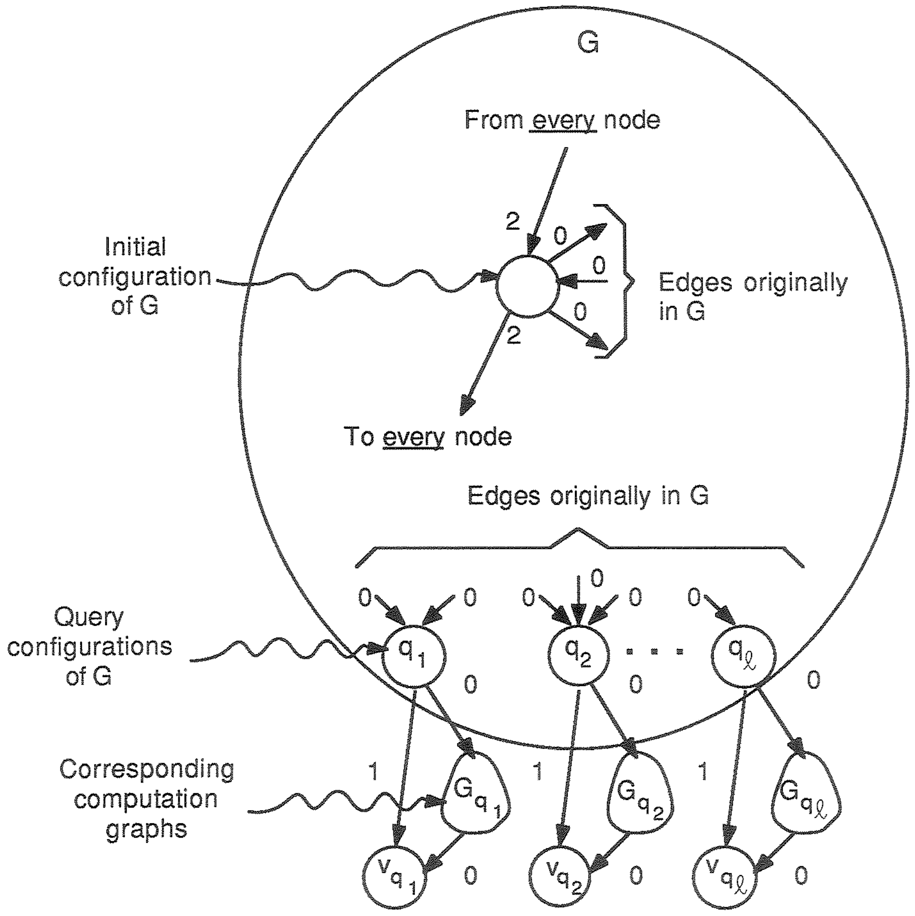


Figure 4.3: Reduction to C.3.

Theorem 2.1: DL $\subseteq$ NL,Co-NL $\subseteq$ D^L $\subseteq$ $\mathcal{L}_{hd}$(NL) $=$ $A\Sigma_2^L$ $=$ $DL^{NL[\log]}$ $\subseteq$ DL^{NL} $\subseteq$ $\cup_{k\geq 2}\Sigma_k^L$ $\subseteq$ $A\Sigma_{\log}^L$ $\subseteq$ P$\cap$DSPACE($\log^2$ n). (See Figure 2.1.)

Proof: The fact that $\mathcal{L}_{hd}$(NL) $=$ $A\Sigma_2^L$ $=$ $\cup_{k\geq 2}A\Sigma_k^L$ was recently shown in [15]. The fact that $DL^{NL[\log]}$$=$$A\Sigma_2^L$ is shown in the next section. The fact that $\cup_{k\geq 2}\Sigma_k^L$ $\subseteq$ $A\Sigma_{\log}^L$ was shown in Lange [14]. The fact that $A\Sigma_{\log}^L$ $\subseteq$ P$\cap$DSPACE($\log^2$ n) follows from the result attributed to Borodin in [3]. The remaining facts are either obvious from the definitions or very easy to derive. □

In the lower bound proofs that follow, we assume that all machines are of a certain canonical form. First of all, since all of the machines we consider operate under a fully space constructible space bound, we assume without loss of generality that all computations halt. Also, when considering alternation bounded ATMs, or OTMs making a limited number of oracle calls, we assume without loss of generality that these bounds are actually reached on all computations (e.g., all 2 alternation bounded ATMs make exactly one alternation on every computation path).

3. A deterministic characterization of the alternating logspace hierarchy

Lange, Jenner and Kirsig in [15] show that $A\Sigma_2^L$$=$$\mathcal{L}_{hd}$(NL). Since the latter is closed under complementation, [15] establishes the surprising result that the alternating logspace hierarchy collapses. In this section we provide a short proof that $DL^{NL[\log]}$$=$$A\Sigma_2^L$. Since $DL^{NL[\log]}$ is clearly closed under complementation, this yields a succinct proof that the alternating logspace hierarchy collapses. More importantly, it establishes a deterministic computational model for $A\Sigma_2^L$, $\mathcal{L}_{hd}$(NL), and the alternating logspace hierarchy.

In [21], we gave a characterization of the alternating logspace hierachy in terms of OTMs. (See also [13].) In particular, [21] shows that a language is in $A\Sigma_2^L$ iff it can be accepted by an O(log n) space bounded nondeterministic OTM that makes a single oracle call to an NL oracle. We will utilize this characterization in showing that $DL^{NL[\log]}$$\subseteq$$A\Sigma_2^L$. The main idea behind the proof that $A\Sigma_2^L$$\subseteq$$DL^{NL[\log]}$ is similar to that of Lemma 4.1 in [15]. In fact, the $A\Sigma_2^L$-complete problem utilized in [15] can be shown to be in $DL^{NL[\log]}$ using essentially the same strategy; however, we feel that the following proof is more direct and succinct.

Theorem 3.1: $DL^{NL[\log]}$$=$$A\Sigma_2^L$.

Proof: We first show that $DL^{NL[\log]}$$\subseteq$$A\Sigma_2^L$. Let L be an arbitrary language in $DL^{NL[\log]}$. Then there is a language A$\in$NL and an O(log n) space bounded deterministic OTM M_1 such that L$=$L($M_1{}^A$), and M_1 makes $\lceil$c*log n$\rceil$ oracle calls, for some constant c, on any input of length n. Let M_2 be an O(log n) space bounded NTM that accepts A. We now describe an O(log n) space bounded nondeterministic OTM M that accepts L using an NL oracle. M will make only a single oracle call during the course of a computation. M on input x will guess an m-bit binary number z, where m is the number of oracle calls M_1 makes on inputs of length |x|. M will then simulate M_1 on x, using z to determine the outcome of the oracle calls as follows: the i^{th} query is assumed to have an outcome of no iff the i^{th} bit (from the left) of z is 0. For all bits of z that are 1, M verifies the yes response by simulating M_2 on x#y, where y is the string on the query tape of M_1 when the oracle call is simulated. If M fails to reach an accepting state of M_1 by this procedure, M rejects. It is not hard to see that the largest z for which all 1 bits can be verified represents the correct computation of ${M_1}^A$. M therefore asks its oracle whether there is a $z'>z$ for which the above procedure also leads to an accepting state of M_1. This question can clearly by answered by an NL oracle. M accepts iff the oracle response is no. Hence, from results in [21], we have that $L \in A\Sigma_2^L$.

We now show that $A\Sigma_2^L \subseteq DL^{NL[\log]}$. Let L be an arbitrary language in $A\Sigma_2^L$. There exists an O(log n) space bounded 2 alternation bounded ATM M such that L(M)=L. We will construct an O(log n) space bounded deterministic OTM M′ that makes no more than c*log n oracle calls (for some constant c) on any input of length n and a set A∈NL such that $L(M'^A)=L$. Let A = {x#0\$i : at least i distinct universal configurations of M are reachable via only existential moves on input x} ∪ {x#1\$i : on input x, there are at least i universal configurations of M reachable via only existential moves, and from each of these i configurations, some rejecting configuration is reachable}. Clearly, A∈NL. For any input x to M, M′ first determines how many universal configurations are reachable via only existential moves. This number i can clearly be determined by using a binary search and consulting an oracle for A O(log |x|) times. Now M rejects x iff x#1\$i∈A. Therefore, by making one more oracle call, M′ can determine whether x∈L. □

Since $DL^{NL[\log]}$ is clearly closed under complementation, this suffices as a succinct proof for the main result of [15]. The characterization can be also used to prove other identities that otherwise might seem unlikely. For example, we also have $NL^{NL[1]}=NL^{DL^{NL[1]}[1]}=NL^{DL^{NL[\log]}[1]}$.

4. Directed graph problems

In this section, we examine the complexity of a number of problems involving directed graphs with unary edge weights. Problems involving directed graphs are very central to the study of computer science. For example, in [1], Aho, Hopcroft, and Ullman devote an entire chapter of a textbook on data structures to the discussion of directed graph problems and algorithms. A substantial portion of this chapter in [1] examines minimum cost path problems. In order to gain a better understanding of the complexity of these problems, we give in Subsection 4.1 completeness results for several problems related to computing the minimum cost path in a graph with unary edge weights. In Subsection 4.2, we apply the techniques employed in Subsection 4.1 to the related problems of finding the minimum length path, the eccentricity of a vertex, the radius of a vertex, and the center of a graph. See [1] for a discussion of these problems, and [4], where parallel algorithms are mentioned for many of the same problems using unary edge weights. In Subsection 4.3, we explore analogies between the class $DL^{NL[\log]}$ and the classes P^{NP} and $P^{NP[\log]}$, comparing our results with those of Krentel [11].

4.1. Minimum cost path problems

In this subsection, we give completeness results for a number of decision problems related to the problem of finding the minimum cost path in a directed graph when edge costs are given in unary. Let N denote the natural numbers, and let ω be a special symbol denoting "plus" infinity. In general, an instance of a problem consists of a graph G with n vertices and a list of triples $<u_i,v_i,k_i>$, $1 \leq i \leq m$, where u_i and v_i are vertices in G, and $k_i \in N \cup \{\omega\}$. Associated with each edge (u,v) in G will be a cost expressed in unary, denoted *cost(u,v)*. (We also extend this notation so that cost(σ) denotes the cost of a path σ.) Let MCP(u_i,v_i) be defined as the value of the minimum cost path from u_i to v_i in G, or ω if no path from u_i to v_i exists. All of the problems we consider have to do with verifying that certain minimum cost paths in a graph meet certain requirements. In particular, we consider the following questions:

A. $\forall i$ MCP(u_i,v_i)$\leq k_i$?

B. $\forall i$ MCP(u_i,v_i)$=k_i$?

C. $\forall i$ MCP(u_i,v_i) mod $k_i = 0$? (We define k mod ω = k and ω mod ω = 0 for all $k \in N$; ω mod k is undefined for $k \in N$.)

Now for each of the above questions, we consider three problems depending upon the value of m (i.e., the number of triples allowed in the list):

1. $m=1$ (i.e., single source, single sink minimum cost path);
2. m is a variable, $1 \leq m \leq n^2$ (i.e., arbitrary number of pairs -- this includes single source minimum cost path); or
3. $m=n^2$ (i.e., all pairs minimum cost path).

Thus, we consider 9 problems, A.1, A.2, ..., C.3.

In showing some of our lower bounds, we make use of the *graph accessibility problem* (GAP), which is the problem of determining, from a directed graph G and vertices u and v, whether there is a path from u to v in G. This problem was shown to be NL-complete in [10]. We also use a number of "generic" reductions. These reductions make use of the *computation graph* of an O(log n) space bounded machine M on a given input x. The computation graph has for its vertex set the set of all possible configurations of M on x. Its edge set then gives all possible moves between configurations. Clearly, the computation graph can be deterministically constructed from M and x using only logarithmic space [10].

We now examine the problems A.1, A.2, and A.3. It is easy to show that A.1 and A.2 are computationally equivalent to GAP; i.e., they are NL-complete. The proof that A.3 is NL-complete is only slightly more difficult. Hence, we have:

Theorem 4.1: A.1, A.2, and A.3 are NL-complete.

Proof: We only consider problem A.3. A.3 is clearly in NL. We will show, using a reduction from GAP, that A.3 is also NL-hard, and therefore NL-complete. Let $<G,u,v>$ be an instance of GAP, where $G=(V,E)$. Let $G'=(V,E')$, where $E' = E \cup \{(x,u) : x \in V\} \cup \{(v,x) : x \in V\}$. Let x and y be arbitrary vertices in V. If there is a path from u to v in G, then there is clearly a path from x to y in G'. Thus, G' is strongly connected if there is a path from u to v in G. Conversely, if G' is strongly connected, there is a simple path from u to v in G'. Since all edges in $E' \setminus E$ either enter u or leave v, all of the edges in the simple path from u to v are in E. Hence, G' is strongly connected iff there is a path from u to v in G. If we now let all edges in G' have cost 0, then $MCP(x,y) \leq 0$ for all $x,y \in G'$ iff G' is strongly connected iff there is a path from u to v in G. □

Notice that in order to decide A.1, A.2, and A.3, it is not necessary to evaluate the exact cost associated with the minimum cost path. Thus, it might seem reasonable that B.1, B.2, and B.3 might be more difficult problems. We give positive evidence of this in the case of B.1 and B.2 by showing them to be D^L-complete; however, B.3 remains NL-complete. This result is somewhat surprising, but a similar result can be shown with respect to the transitive closure problem (for unweighted graphs); i.e., the problem of verifying whether some subset of bits in a potential transitive closure of a Boolean matrix can be shown to be D^L-complete, but the problem of verifying an entire transitive closure matrix can be shown to be NL-complete. The proofs of these facts are similar to those that follow.

Theorem 4.2: B.1 and B.2 are D^L-complete.

Proof: We first show B.1 to be D^L-hard. Let L be an arbitrary language in D^L. Then $L=L_1 \cap L_2$, where $L_1 \in NL$ and $L_2 \in co\text{-}NL$. Let M_1 and M_2 be NTMs using logarithmic space such that $L(M_1)=L_1$ and $L(M_2)=co\text{-}L_2$. Let x be an input to M_1 and M_2. We now construct the computation graphs G_1 and G_2 of M_1 and M_2, respectively, on x, such that G_1 and G_2 have disjoint vertex sets. We then join G_1 and G_2 by adding edges from the accepting configurations in G_1 to the starting configuration in G_2. Also, we add a new node v and edges from the accepting configurations in G_2 to v. Finally, we assign a cost of zero to all existing edges and add an edge with cost 1 from the starting configuration in G_2 to v. See Figure 4.1. Clearly, the minimum cost path from the starting configuration in G_1 to v has cost 1 iff $x \in L(M_1) \setminus L(M_2) = L_1 \cap L_2 = L$.

We now show B.2 to be in D^L. Let A.2$'$ be A.2 modified such that $<G, \{u_i,v_i,k_i : 1 \leq i \leq m\}> \in$ A.2$'$ iff $\forall i$ MCP$(u_i,v_i)<k_i$. Clearly A.2$' \in$NL. Now any instance of B.2 can be expressed as an instance of A.2$\cap$co-A.2$'$; therefore, B.2$\in D^L$. It now follows that both B.1 and B.2 are D^L-complete. □

Theorem 4.3: B.3 is NL-complete.

Proof: The fact that B.3 is NL-hard follows as a corollary to the proof of Theorem 4.1. Hence, we need only show that B.3 is in NL. Let G=(V,E) be an arbitrary directed graph, with cost(u,v) denoting the cost of edge (u,v). (If edge (u,v) does not exist, cost(u,v)=ω.) Suppose for each ordered pair $<u,v>$ of vertices in V, we are given a $k_{uv} \in N \cup \{\omega\}$. We give the following algorithm to verify that each k_{uv}=MCP(u,v):

Step 1. $\forall u,v \in V$ verify that MCP$(u,v) \leq k_{uv}$.

Step 2. $\forall u \in V$ verify that $k_{uu}=0$.

Step 3. $\forall (u,v) \in E$ verify that $k_{uv} \leq$ cost(u,v).

Step 4. $\forall u,v,w \in V$ verify that $k_{uv} \leq k_{uw}+k_{wv}$.

From Theorem 4.1, Step 1 can be carried out in nondeterministic logspace, and Steps 2, 3, and 4 can clearly be carried out in deterministic logspace. Furthermore, the algorithm can clearly accept any correct input. Suppose the algorithm accepts some input in which some entry k_{uv} is incorrect. From Step 1, MCP$(u,v)<k_{uv}$. Let σ be a path from u to v with cost less than k_{uv}, and assume without loss of generality that σ is the shortest path in G (in terms of the number of edges) with a lower cost than that proposed in the input. From Step 2, σ has at least one edge. Let w be the vertex following u in σ. From Step 3, cost$(u,w) \geq k_{uw}$. Let σ' be that portion of σ beginning with w. By our choice of σ, cost$(\sigma') \geq k_{wv}$. From Step 4, $k_{uv} \leq k_{uw}+k_{wv} \leq$ cost(u,w)+cost(σ')=cost(σ) -- a contradiction. The theorem now follows.

□

Now in all of the problems examined thus far, it has not been necessary to compute the exact value associated with the minimum cost path. This computation turns out to be necessary for C.1, C.2, and C.3, which are all complete for $A\Sigma_2^L$. Now since $A\Sigma_2^L = DL^{NL[\log]}$, this result might be expected given the results in [11, 17]. In [17], Papadimitriou showed that the uniquely optimal travelling salesperson problem is P^{NP}-complete. Krentel points out in [11] that the inherent complexity in this problem arises from the necessity of computing the optimal tour, and that the problem of determining whether the length of the optimal tour is 0 mod k is P^{NP}-complete for the same reason. Now since A.1 is NL-complete (rather than NP-complete, like the travelling salesperson problem), one might expect C.1 to be complete for DL^{NL}. However, it turns out that only O(log n) oracle calls are needed. Hence, the class

$DL^{NL[\log]}$ (or equivalently $A\Sigma_2^L$) may be considered to be in some sense analogous to P^{NP}. (We discuss this relationship in more detail in Subsection 4.3.) Note, however, that although $DL^{NL[\log]}$ has a natural characterization in terms of optimization problems (see Subsection 4.3), the problems shown in [7, 13, 21, 22] to be complete for $A\Sigma_2^L$ do not appear to be optimization problems; rather, the problems in [7, 21, 22] have to do with finding particular types of strongly connected components in directed graphs. Thus, the equivalence of $DL^{NL[\log]}$ to $A\Sigma_2^L$ unifies two groups of problems which appear to be quite different. Furthermore, the alternate characterization of $A\Sigma_2^L$ provides us with another tool for showing completeness results. We take advantage of this fact in the theorems that follow. The characterization used in each proof is utilized in the statement of the respective theorem.

Theorem 4.4: C.1 is $DL^{NL[\log]}$-complete.

Proof: We first show C.1$\in DL^{NL[\log]}$. Given an instance of C.1, let n be the number of vertices and c be the largest cost of any edge. Then if $MCP(u_i,v_i)$ is finite, it can be no more than c*n. We therefore use a binary search to find the smallest k such that $MCP(u_i,v_i)\leq k$. Each k can be written in log(c*n) bits, and at most log(c*n) calls to an oracle for A.1 need to be made. Once $MCP(u_i,v_i)$ is found, $MCP(u_i,v_i)$ mod k is easily computable in DL. Thus C.1$\in DL^{NL[\log]}$.

We now show C.1 to be $DL^{NL[\log]}$-hard. Let M be an arbitrary O(log n) space-bounded deterministic OTM which queries its oracle f(n) times in any computation on an input of length n, where $f(n)=\lceil c*\log n\rceil$ and is fully space constructible, and let A be an arbitrary language in NL. Let x be an arbitrary input string of length n. We can construct in deterministic logspace a tree T of depth f(n) such that the root node is the configuration of M at its first oracle query, the left child of a node p is the configuration of M at its next query after a "no" response from the oracle at configuration p, and the right child of p is the configuration of M at its next query after a "yes" response from the oracle at p. The leaves of T are the halting configurations. We now construct a graph G from T as follows. First assign to all left-hand edges a cost $2^{f(n)-d+1}$, where d is the depth of the head of the edge. Let M′ be an O(log n) space-bounded nondeterministic TM that accepts A. We now insert between each node p in T and its right-hand child the computation graph of M′ on the query tape in configuration p such that from all accepting configurations of M′ there is an edge to the right-hand child of p. See Figure 4.2. Now add an edge with cost 0 from every accepting configuration of M to a new vertex v, and add an edge with cost 1 from every rejecting configuration of M to v. Give a cost of 0 to every edge which has not yet been given a cost.

We now claim that the minimum cost path from the initial configuration to v represents the computation of M^A on x. To see this, first observe that from any query node p, if the correct response from A is "no," then there can be no path from p to v through the right-hand child of p. Furthermore, if the correct response from A is "yes," then the minimum cost path from p to v must be through the right-hand child of p because the cost of this path is at most $\sum_{i=d+1}^{f(n)} 2^{f(n)-i+1}+1 < 2^{f(n)-d+1}$, the cost of the edge to the left-hand child of p. Thus, the minimum cost path must clearly represent the computation of M^A on x. Furthermore, the minimum cost path is even-valued iff it passes through an accepting configuration. Therefore, the minimum-cost path is even-valued iff M^A accepts x. □

Theorem 4.5: C.2 and C.3 are $A\Sigma_2^L$-complete.

Proof: We will show that $C.2 \in A\Sigma_2^L$ and C.3 is $A\Sigma_2^L$-hard. The result will then follow.

In order to show that $C.2 \in A\Sigma_2^L$, we will describe an O(log n) space bounded 2 alternation bounded ATM M that accepts all instances which are not in C.2. From [15], it will then follow that $C.2 \in A\Sigma_2^L$. M operates as follows. First, M guesses an i, $1 \leq i \leq m$. If $k_i = \omega$, then M guesses a path from u_i to v_i and accepts if successful. Otherwise, M guesses as to whether a path exists from u_i to v_i. We then have the following two cases, depending upon the guess made by M.

Case 1: M guesses that there is a path from u_i to v_i. M will now attempt to verify this guess and to show that $MCP(u_i,v_i) \bmod k_i \neq 0$. M first guesses a path from u_i to v_i and computes the cost k. If M fails to find a path with cost k, where $k \bmod k_i \neq 0$, it rejects. If such a path is found, M enters a universal state. M will now attempt to show that $MCP(u_i,v_i)=k$. It does this by computing the cost of each path from u_i that contains no more than n edges. If some path leads to v_i and has cost $< k$, M rejects; otherwise, M accepts.

Case 2: M guesses that there is no path from u_i to v_i. M will enter a universal state, and follow all paths from u_i no longer than n. If any path reaches v_i, M will enter a rejecting state; thus, if no path ends in v_i, M accepts its input.

Clearly, M accepts all instances for which there is an i such that $MCP(u_i,v_i) \bmod k_i \neq 0$.

We will now show that C.3 is $A\Sigma_2^L$-hard. Let L be an arbitrary language in $A\Sigma_2^L$. From [21], there is an O(log n) space-bounded nondeterministic OTM M and a set $A \in NL$ such that $L(M^A)=L$. Furthermore, we can assume that in any computation, M makes exactly one oracle call and accepts iff the response is "no". We can construct, in deterministic logspace, the computation graph G of M on a given input x, such that the query configurations have no outgoing edges. We assign a cost of 0 to every edge in G. Now from every query configuration q in G, we add an edge having cost 1 to a new vertex v_q. Let M' be an O(log n) space-bounded nondeterministic TM such that $L(M')=A$. We now construct, for every query configuration q in G, the computation graph G_q of M' on $x\#y$, where y is the string on the query tape in q. We assign a cost of 0 to every edge in G_q. We now combine G with all of the G_q's by adding edges with cost 0 from each query configuration q to the initial configuration of G_q, and from each accepting configuration of G_q to v_q. Finally, we add edges having cost 2 from the initial configuration of M to every vertex in the graph, and from every vertex in the graph to the initial configuration. Call the resulting graph G′. See Figure 4.3.

Now let k_i be defined as follows. If u_i is the initial configuration of M and $v_i = v_q$ for some query configuration q, then $k_i = 2$; otherwise, $k_i = 1$. We now claim that there is an i, $1 \leq i \leq n^2$, such that $\mathrm{MCP}(u_i, v_i) \bmod k_i \neq 0$ iff $x \in L$. Suppose such an i exists. Since G′ is clearly strongly connected, it must be the case that $k_i = 2$; thus, u_i must be the initial configuration of M, and v_i must be v_q for some query configuration q. Since there is an edge from u_i to v_i with cost 2, $\mathrm{MCP}(u_i, v_i)$ must be 1; thus, no edges having cost 2 are in the minimum cost path. It is therefore easy to see that configuration q is reachable in a computation of M on x, and that the response from the oracle call in q must be "no". Therefore, $x \in L$. Now suppose, conversely, that $x \in L$. It is again easily seen that $\mathrm{MCP}(u_i, v_i) = 1$, where u_i is the initial configuration of M and $v_i = v_q$ for some query configuration q. Therefore, since $A\Sigma_2^L$ is closed under complementation, C.3 is $A\Sigma_2^L$-hard. □

4.2. Applications to related problems

In this subsection, we apply the techniques used in the previous subsection to several related problems. These problems include minimum length path problems (i.e., the problems considered in the previous section in which all edge weights are 1), as well as problems involving the eccentricity of a vertex, the radius of a vertex, and the center of a graph.

Let v be a vertex in a directed weighted graph $G=(V,E)$. The *eccentricity* of v is defined in [1] to be $\max_{u \in V}\{\mathrm{MCP}(u,v)\}$. A metric closely related to the eccentricity of a vertex v is the *radius* of v, defined as $\max_{u \in V}\{\mathrm{MCP}(v,u)\}$. As was done above in defining the minimum length path problems, we can modify in an obvious way the problems A.1, A.2, ..., C.3, from the previous subsection to ask questions about the eccentricities (or radii) of vertices. For example, A.1 can be modified to ask whether a given vertex has eccentricity $\leq$ k; A.2 can be modified to ask whether a given list of vertices $v_1, \ldots, v_m$ have eccentricities $\leq k_1, \ldots, k_m$, etc. It can be shown that whether we are talking about minimum length path, eccentricity, or radius, the complexities are the same as the corresponding problem from Subsection 4.1 except for those corresponding to problem B.3.

Consider the problem B.3 restricted to edge weights of 1. Recall that in the unrestricted case, the NL upper bound was somewhat surprising. We now show an even more surprising result, that when edge weights are restricted to be 1, the problem can be solved in deterministic logspace. Consider the following algorithm.

```
for each vertex u, verify that $k_{uu}=0$
for each $k_{uv}$, verify that either $k_{uv}<n$ or $k_{uv}=\omega$
for $i=1,\ldots,n-1$
        for each $k_{uv}=i$, verify that there is an edge (u,w) such that $k_{wv}=i-1$
        for each edge (u,v) and each $k_{vw}=i-1$, verify that $k_{uw}\leq i$
```

Clearly, the above algorithm requires O(log n) space. Furthermore, it is not hard to show that it successfully terminates iff all k_{uv} are correct. We therefore have the following theorem.

Theorem 4.6: The problem B.3 restricted to cases in which all edge weights are 1 can be solved in deterministic logspace.

We now consider the problems of verifying the eccentricity (or radius) of every vertex in a graph (i.e., the problems corresponding to B.3). These problems are easily seen to be in D^L. In the following theorem, we show the problems to be D^L-complete.

Theorem 4.7: Given a directed graph G with m vertices and edge weights in unary, and an integer k_i, $1\leq i\leq m$, for each vertex u_i, the problem of deciding whether, for all vertices u_i, the eccentricity (or radius) of u_i is k_i is D^L-complete.

Proof: Let $<G,u,v>$ be an instance of GAP, and let $<G',u',v'>$ be an instance of co-GAP. We will construct a graph G_1 with unary edge weights such that the eccentricity (radius) of every vertex in G_1 is 1 iff there is a path from u to v in G and there is no path from u′ to v′ in G′. First, let G″ be a copy of G′ such that for each vertex w′ in G′, the vertex w″ in G″ is isomorphic to w′. We now combine G, G′, and G″ into a graph G_1 as follows. First, add edges from v (v′, v″) to every vertex in G (G′, G″, resp.) and from every vertex in G (G′, G″) to u (u′, u″, resp.). See Figure 4.4. Now add the edges (v, u′), (v, u″), (u′, u), (u″, u), (v′, u), and (v″, u). Assign to all edges currently in G_1 a cost of 0. Finally, add the edges (u′, v′) and (u″, v″) with costs of 1. It is now a straightforward matter to show that all vertices have eccentricity (radius) 1 iff there is a path from u to v in G but no path from u′ to v′ in G′. □

A related and perhaps more interesting problem is that of finding the center of a graph. A *center* of a graph G is defined defined in [1] as any vertex with minimum eccentricity. (In [19], a center of a graph is defined as any vertex with minimum radius. Clearly, any two problems that differ only by these definitions can be shown to be equivalent by reversing the direction on all edges.) In what follows, we will show that the problem of determining whether a given vertex v is the center of a given graph G is $DL^{NL[log]}$-complete.

Theorem 4.8: The problem of determining whether a vertex v is a center of a directed graph G is $DL^{NL[log]}$-complete.

Proof: First note that v is not a center of G iff there exist a vertex u and an integer m such that the eccentricity of u is $\leq$ m and the eccentricity of v is $>$ m. Therefore, the complement of the problem is in $A\Sigma_2^L$, so the problem itself is in $A\Sigma_2^L = DL^{NL[\log]}$. We now will show the problem to be $DL^{NL[\log]}$-hard. Consider the construction given in Theorem 4.4. We will modify the graph G constructed there so that v is a center of the resulting graph G′ iff $x \in L(M^A)$. We first add edges with cost 0 from all vertices except v to the starting configuration. We then add edges with cost 0 from every halting configuration to a new node u. Finally, we add edge (u,v) with cost 1 and edge (v,u) with cost 0. See Figure 4.5. Now there is a path in the resulting graph G′ from every vertex to both u and v, but there is no path from u or v to any vertex outside of {u,v}. Hence, only u and v have finite eccentricities. Clearly, the vertex with the highest minimum cost path to either u or v is the starting configuration. By reasoning as in Theorem 4.4, this minimum cost path represents the actual computation of M with oracle A. Therefore, u is a center of G′, and v can be a center iff $x \in L(M^A)$. □

4.3. Analogies from P^{NP} and $P^{NP[\log]}$ to $DL^{NL[\log]}$

In [11], Krentel characterized the classes P^{NP} and $P^{NP[\log]}$ in terms of optimization functions followed by polynomial-time computations. In particular, he defined a *metric* TM as a nondeterministic transducer whose output is defined to be the largest integer that can be written to its output tape in any computation. He then defined OptP to be the set of functions computable by polynomial time bounded metric TMs, and OptP[z] to be the set of functions computable by polynomial time bounded metric TMs whose outputs are bounded by z(n) bits. He was then able to show that for sufficiently "nice" z, $f:\Sigma^* \to \{0,1\}$ is in $P^{NP[z]}$ iff there exist functions $g:\Sigma^* \times N \to \{0,1\}$ in P and $h:\Sigma^* \to N$ in OptP[z] such that $f(x)=g(x,h(x))$.

It is not hard to see that these techniques can be used to show a similar result for $DL^{NL[\log]}$. I.e., let OptL[z] be the set of functions computable by O(log n) space bounded metric TMs whose outputs are bounded by z(n) bits. We then have the following theorem, which can be shown in the same way as Theorems 3 and 4 in [11].

Theorem 4.9: If z is a nondecreasing function such that the function $\phi(1^n)=1^{z(n)}$ is computable in deterministic logspace and $z(n)=O(\log n)$, then $f:\Sigma^* \to \{0,1\}$ is in $DL^{NL[z]}$ iff there exist functions $g:\Sigma^* \times N \to \{0,1\}$ in DL and $h:\Sigma^* \to N$ in OptL[z] such that $f(x)=g(x,h(x))$.

Thus, any language in $DL^{NL[\log]}$ can be recognized by an OptL[log] computation followed by a DL computation. This is somewhat surprising, since the problems shown in [7, 21, 22] have to do with finding particular types of strongly connected components, and hence do not have the appearance of optimization problems. One of the chief reasons the generalizations go through for $DL^{NL[z]}$, when

5. Applications to other hierarchies involving space

Given the fact that the alternating logspace hierarchy collapses, one might wonder if alternating hierarchies based on space bounds other than log n also collapse. A partial answer to this question follows from a result attributed to Borodin in [3], namely, if L is accepted by an S(n) space bounded A(n) alternation bounded ATM with $S(n) \geq \log n$, then L is accepted by an $S(n)*A(n)+S^2(n)$ space bounded deterministic TM. From this, it follows that any polynomial space-bounded k alternation bounded ATM can be simulated by a polynomial space-bounded deterministic TM. However, Borodin's result clearly cannot be applied to $\log^2 n$ space or linear space, for example, to imply a similar collapse. We address this issue by showing that any alternating hierarchy based upon S(n) space-bounded ATMs with $S(n) \geq \log n$ must collapse. This is stated more formally in the following theorem and its corollary. The proof of this theorem again uses the same key ideas as were used in [15], and in fact, generalizes the main result of [15] very succinctly.

Theorem 5.1: If S is a fully space constructible function such that $S(n) \geq \log n$, then for any S(n) space bounded 2 alternation bounded ATM M, there is an S(n) space bounded 2 alternation bounded ATM M′ such that $L(M') = \overline{L}(M)$.

Proof: We construct M′ as follows. M′ first generates in some canonical order the universal configurations of M on a given input x, keeping only one configuration at a time written on its work tape. Since S(n) is fully space-constructible, each configuration can be generated and written down in O(S(n)) space. (Since the input tape remains unchanged, we do not include it in the description of a configuration. We only need a pointer to the tape head position.) For each configuration σ, M′ guesses whether that configuration is reachable by some path in M in which all configurations except σ are existential. If M′ guesses σ to be unreachable, it generates the next configuration in the canonical ordering; otherwise, it guesses a path to σ through only existential states, then from σ through only universal configurations until it reaches a halting state. If M′ fails to reach σ or reaches an accepting state of M, M′ rejects x. Otherwise, σ is counted, and the next configuration is generated. Clearly, the paths described above can be guessed using O(S(n)) space. Since the total number of configurations of M is $n*c^{S(n)}$ for some constant c, the counter can be stored in $\log n + S(n)*\log c = O(S(n))$ bits. After all configurations have been generated, M′ enters a universal state. Suppose the counter at this point is i. M′ now verifies that no more than i universal configurations can be reached by M in its existential phase. M′ can clearly do this in a manner similar to that described above, rejecting if more than i universal configurations are found to be reachable. Clearly, $L(M') = \overline{L}(M)$, and by a tape compression argument, M′ can be made to operate in S(n) space. □

Corollary 5.1: If S is a fully space-constructible function such that $S(n) \geq \log n$, then any S(n) space-bounded k alternation bounded ATM can be simulated by an S(n) space-bounded 2 alternation bounded ATM; i.e., $A\Sigma_k^{S(n)}=A\Sigma_2^{S(n)}$.

Another question one might ask is whether other hierarchies similar to the alternating logspace hierarchy also collapse. Perhaps the most closely related hierarchy is the symmetric complementing logspace hierarchy of Reif [20]. The machine used to define this hierarchy is the complementing Turing machine (CTM). Informally, a *complementing Turing machine* is an NTM M with two types of transitions, *symmetric* transitions, and *complement* transitions. The symmetric transitions must have the property that for any configurations I and I′ of M, if t is a symmetric transition such that $I \vdash_t I'$, then there is a symmetric transition t′ such that $I' \vdash_{t'} I$. For a formal definition of CTMs, see [20]. Acceptance in a CTM is defined inductively as follows. Any accepting configuration is said to lead to acceptance. A nonaccepting configuration I of M leads to acceptance iff either some successor of I via a symmetric transition leads to acceptance, or there is at least one complement transition enabled at I and no successor of I via any complement transition leads to acceptance. A CTM is said to accept its input iff its initial configuration leads to acceptance. Let SL be the set of languages accepted by O(log n) space bounded CTMs that take no complement transitions. (We also call these machines *symmetric* TMs; see also [16].) Let $C\Sigma_f^{SL}$ be the set of languages accepted by O(log n) space bounded CTMs that take at most f(n)-1 complement transitions during the course of a computation. The complementing symmetric logspace hierarchy is then defined to be $\cup_{k\geq 1}C\Sigma_k^{SL}$. It is not hard to see that $C\Sigma_k^{SL} \subseteq A\Sigma_k^{L}$ (see [20]); hence, $\cup_{k\geq 1}C\Sigma_k^{SL} \subseteq A\Sigma_2^{L}$.

Two questions one might ask regarding this hierarchy are

- Is $\mathcal{L}_{hd}(SL) = C\Sigma_2^{SL}$?
- Is $DL^{SL[\log]} = C\Sigma_2^{SL}$?

Clearly, a positive answer to either of the above questions would imply the collapse of the symmetric complementing logspace hierarchy. Although we are unable to resolve either of the above questions or to show the collapse of the hierarchy, we show in this section that $\mathcal{L}_{hd}(SL)=DL^{SL[\log]}\subseteq C\Sigma_2^{SL}$. We then point out why the strategies employed in [15] and Theorem 3.1 of this paper do not appear to extend to the symmetric complementing logspace hierarchy.

We first observe that techniques employed in [21] may be used to show the following theorem, which gives an alternative characterization of $C\Sigma_2^{SL}$ in terms of symmetric OTMs making at most one call to an SL oracle. (In defining symmetric OTMs, the transitions to and from query states are neither symmetric nor complementing.)

Theorem 5.2: $SL^{SL[1]}=C\Sigma_2^{SL}$

Theorem 5.3: $\mathcal{L}_{hd}(SL)=DL^{SL[\log]}$.

Proof: Let $L\in\mathcal{L}_{hd}(SL)$. Then there is a set $B\in SL$ such that for any input x, we can compute, using only O(log n) space, a list of strings $y_1,\ldots,y_{2*n_x}$ such that for all i, if $y_{i+1}\in B$, then $y_i\in B$, and such that $x\in L$ iff for some i, $y_{2*i-1}\in B$ and $y_{2*i}\notin B$. We can clearly use a binary search to find the smallest i such that $y_i\notin B$. This requires only O(log n) space and O(log n) oracle calls to B. Thus, $L\in DL^{SL[\log]}$.

Now let $L\in DL^{SL[\log]}$. Then there is a deterministic O(log n) space bounded OTM M and a set $A\in SL$ such that M makes $\lceil c*\log n\rceil$ oracle calls on any input of length n, and $L=L(M^A)$. Now for an input x, assign to each computation path in M a $\lceil c*\log n\rceil$-bit binary number b such that the ith bit from the left is 1 iff the ith oracle response in the computation is yes. Now consider the following questions:

q_{2*i+1}: Is there a number $j\geq i$ such that in the path associated with j, M accepts and for all k, if the kth bit of j is 1, then $x\#y\in A$, where y is the content of the query tape at the kth query?

q_{2*i+2}: Is q_{2*i+1} true, and is there a number $j>i$ such that in the path associated with j, if the kth bit of j is 1, then $x\#y\in A$, where y is the content of the query tape at the kth query?

We now claim that the above questions can be answered in SL. Both questions involve guessing a number j of logarithmic length (reversible), comparing it with i (deterministic), and verifying a series of undirected graph accessibility problems which are generated deterministically. Using techniques from [16], both questions can be seen to be in SL. Furthermore, it is not hard to see that for all i, if q_{i+1} is true, then q_i is true, and that there is an i such that q_{2*i-1} is true and q_{2*i} is false iff $x\in L(M^A)$. Therefore, $L\in\mathcal{L}_{hd}(SL)$. □

Theorem 5.4: $DL^{SL[\log]}\subseteq SL^{SL[1]}$.

Proof: Consider the construction of the machine M given in Theorem 3.1. We will show that if M_2 is symmetric, then M can be made to be symmetric. M must be able to guess a number z of logarithmic length (reversible), simulate a deterministic TM (deterministic), and simulate a symmetric machine (symmetric). Thus, using techniques from [16], M can be made to be symmetric. Furthermore, it follows from the proof of Theorem 5.3 that the oracle required by M is in SL. This completes the proof. □

At this time, we would like to be able to show that either $C\Sigma_2^{SL}\subseteq\mathcal{L}_{hd}(SL)$ or $C\Sigma_2^{SL}\subseteq DL^{SL[\log]}$. Either of these containments would imply the equality of all three sets and the collapse of the symmetric complementing logspace hierarchy. However, an important feature of both the proof that $A\Sigma_2^{L}\subseteq\mathcal{L}_{hd}(NL)$ given in [15] and the proof of Theorem 3.1 is the ability of an NTM to count guesses. For example, in the proof of Theorem 3.1, the NTM tries to find i reachable configurations. It can do this by generating configurations, and for each configuration, guess whether it is reachable. If it guesses "yes", it verifies that the configuration is reachable and counts it; otherwise, it doesn't count it. Now i is potentially large enough that there will not be room enough to store all i reachable configurations. Therefore, if this algorithm were to be implemented on a symmetric machine, the machine would not be able to detect when reversing its moves whether a particular configuration had been counted. Hence, in attempting to reverse a move, it might not "uncount" a configuration it had previously counted. This would allow it to count the configuration arbitrarily many times. We do not yet see any way around this difficulty, even at higher levels of the hierarchy.

Acknowledgment: We would like to thank Lane Hemachandra for his comments concerning an earlier version of this paper.

References

[1] Aho, A., Hopcroft, J., and Ullman, J., *Data Structures and Algorithms,* (Addison-Wesley, Reading, Mass., 1983).

[2] Bentley, J., Ottmann, T., and Widmayer, P., The Complexity of Manipulating Hierarchically Defined Sets of Rectangles, in: *Advances in Computing Research 1* (JAI Press Inc., 1983), 127-158.

[3] Chandra, A., Kozen, D. and Stockmeyer, L., Alternation, *JACM 28,* 1 (January 1981), pp. 114-133.

[4] Cook, S., A Taxonomy of Problems with Fast Parallel Algorithms, *Information and Control 64* (1985), 2-22.

[5] Garey, M. and Johnson, D., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* (W. H. Freeman and Company, San Francisco, 1979).

[6] Hopcroft, J. and Ullman, J., *Introduction to Automata Theory, Languages, and Computation,* (Addison-Wesley, Reading, Mass., 1979).

[7] Howell, R., Rosier, L., Huynh, D., and Yen, H., Some Complexity Bounds for Problems Concerning Finite and 2-Dimensional Vector Addition Systems with States, *Theoret. Comp. Sci. 46* (1986), 107-140.

[8] Howell, R., and Rosier, L., Completeness Results for Reachability, Containment, and Equivalence with Respect to Conflict-Free Vector Replacement Systems, to be presented at the *14th International Colloquium on Automata, Languages, and Programming,* July, 1987, Karlsruhe, F.R.G. Also Rep. 86-21, The University of Texas at Austin, Austin, Texas, 78712, 1986.

[9] Huynh, D., A Simple Proof for the Σ_2^P Upper Bound of the Inequivalence Problem for Semilinear Sets, *Elektronische Informationsverarbeitung und Kybernetik 22* (1986), pp. 147-156.

[10] Jones, N., Space-Bounded Reducibility Among Combinatorial Problems, *J. of Computer and System Sciences 11* (1975), 68-75.

[11] Krentel, M., The Complexity of Optimization Problems, to appear in *J. of Computer and System Sciences.* A preliminary version was presented at the *18th Ann. ACM Symp. on Theory of Computing.*

[12] Ladner, R., and Lynch, N., Relativization of Questions about Log Space Computability, *Math. Systems Theory 10* (1976), 19-32.

[13] Lange, K., Two Characterizations of the Logarithmic Alternation Hierarchy, *Proc. of the 12th Symp. of Math. Foundations of Comput. Science* (1986), 518-526.

[14] Lange, K., Decompositions of Nondeterministic Reductions, *Proc. of the 13th International Colloquium on Automata, Languages, and Programming* (1986), 206-214.

[15] Lange, K., Jenner, B., and Kirsig, B., The Logarithmic Alternation Hierarchy Collapses: $A\Sigma_2^{\mathcal{L}} = A\Pi_2^{\mathcal{L}}$, to be presented at the *14th International Colloquium on Automata, Languages, and Programming,* July, 1987, Karlsruhe, F.R.G.

[16] Lewis, H., and Papadimitriou, C., Symmetric Space-Bounded Computation, *Theoret. Comput. Sci. 19* (1982), 161-187.

[17] Papadimitriou, C., On the Complexity of Unique Solutions, *JACM 31* (1984), 392-400.

[18] Papadimitriou, C., and Yannakakis, M., The Complexity of Facets (and Some Facets of Complexity), *J. of Computer and System Sciences 28* (1984), 244-259.

[19] Preparata, F., and Yeh, R., *Introduction to Discrete Structures for Computer Science and Engineering,* (Addison-Wesley, Reading, Mass., 1973).

[20] Reif, J., Symmetric Complementation, *JACM 31* (1984), 401-421.

[21] Rosier, L. and Yen, H., Logspace Hierarchies, Polynomial Time and the Complexity of Fairness Problems Concerning ω-Machines, *Proceedings of the 3rd Annual Symposium on Theoretical Aspects of Computer Science,* LNCS 210 (1986), pp. 306-320. To appear in *SIAM J. Comput.*

[22] Rosier, L. and Yen, H., On the Complexity of Deciding Fair Termination of Probabilistic Concurrent Finite-State Programs, *Proceedings of the 13th International Colloquium on Automata, Languages and Programming,* LNCS 226 (1986), 334-343.

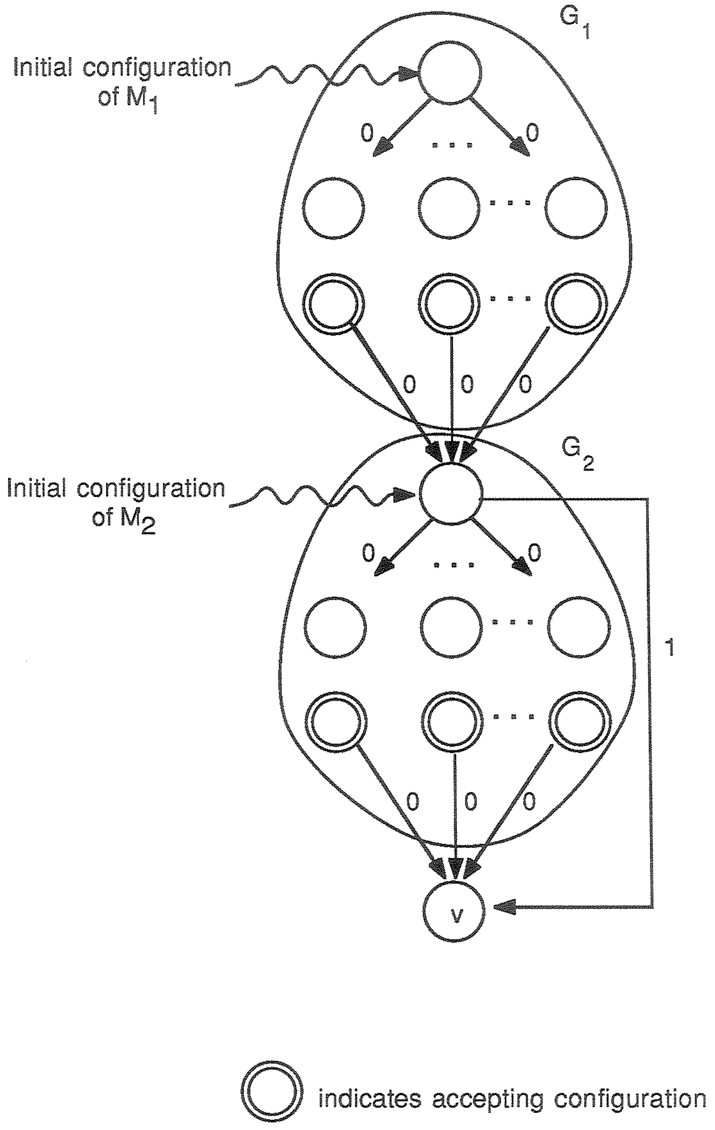
[23] Ruzzo, W., Simon, J., and Tompa, M., Space-Bounded Hierarchies and Probabilistic Computations, *J. of Computer and System Sciences 28* (1984), 216-230.

[24] Stockmeyer, L., The Polynomial-Time Hierarchy, *Theoret. Comp. Sci. 3* (1977), 1-22.

[25] Tompa, M., personal communication.

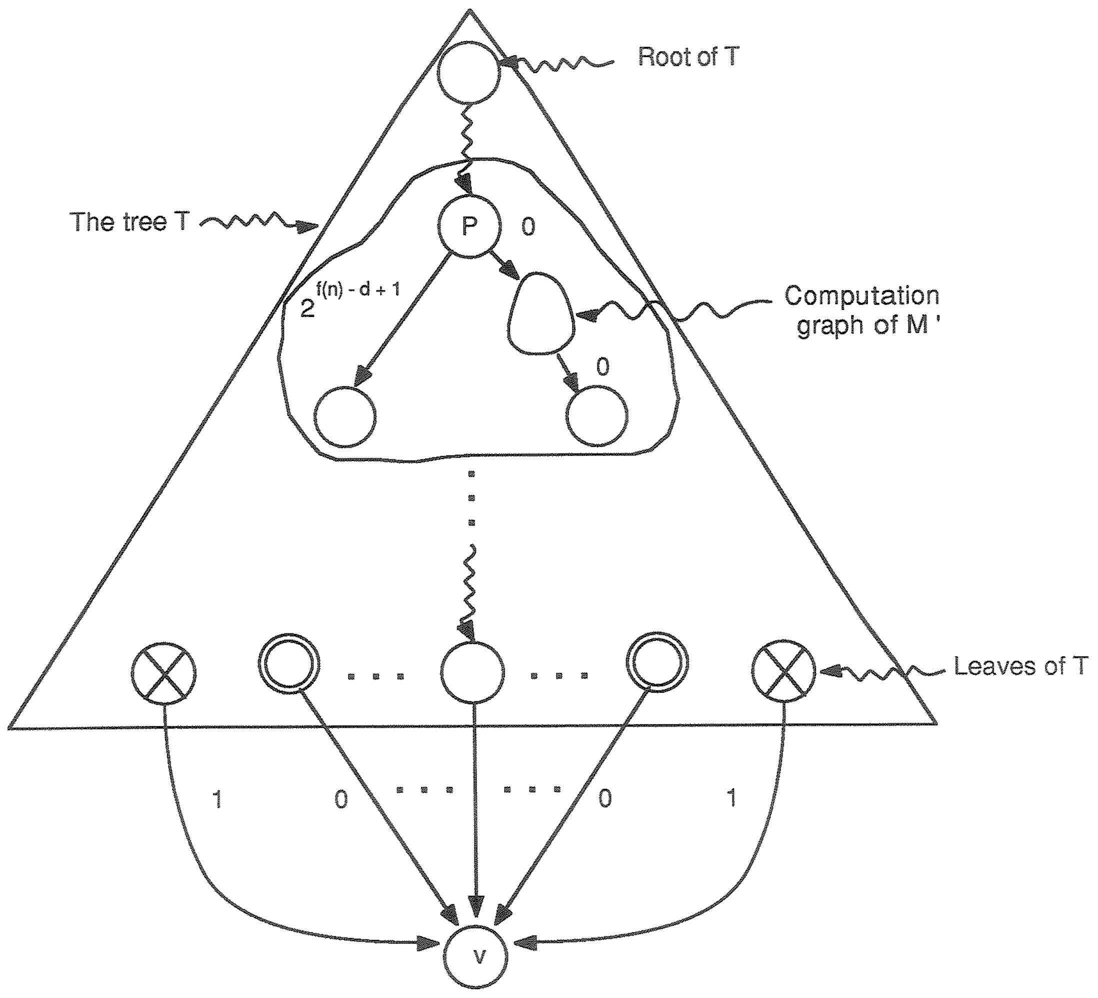
$$\begin{tikzcd}
& \mathrm{PTIME} \cap \mathrm{DSPACE}(\log^2 n) & \\
& A\Sigma^{L}_{\log} \arrow[u] & \\
& \bigcup_{k\geq 2} \Sigma^{L}_{k} \arrow[u] & \\
& \mathrm{DL}^{\mathrm{NL}} \arrow[u] & \\
& L_{hd}(\mathrm{NL}) = A\Sigma^{L}_{2} = \mathrm{DL}^{\mathrm{NL[log]}} \arrow[u] & \\
& D^{L} \arrow[u] & \\
\text{Co-NL} \arrow[ur] & & \mathrm{NL} \arrow[ul] \\
& \mathrm{DL} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 2.1: Inclusions among some sublinear space complexity classes.
(Each directed line "$\longrightarrow$" should be read as " $\subseteq$ ".)



G_1
Initial configuration of M_1
0
0
0
0
0
G_2
Initial configuration of M_2
0
0
1
0
0
0
v
indicates accepting configuration

Figure 4.1: Reduction to B.1.



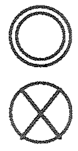

indicates accepting configuration
indicates rejecting configuration

Figure 4.2: Reduction to C.1.

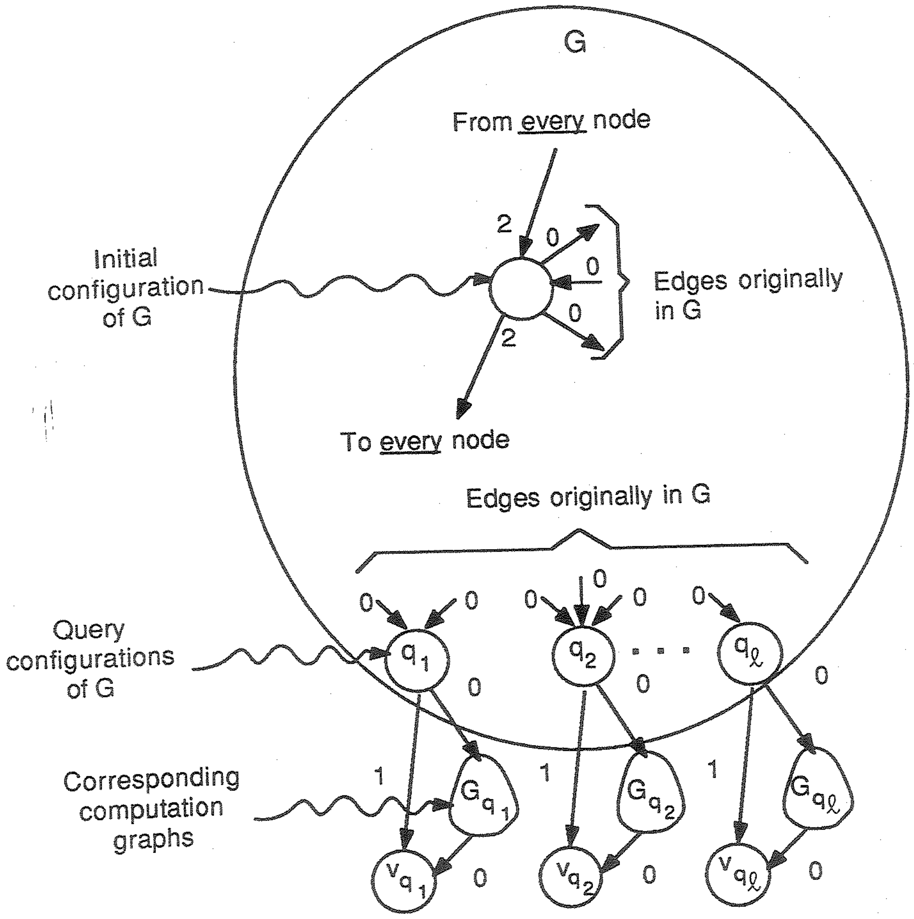


Figure 4.3: Reduction to C.3.

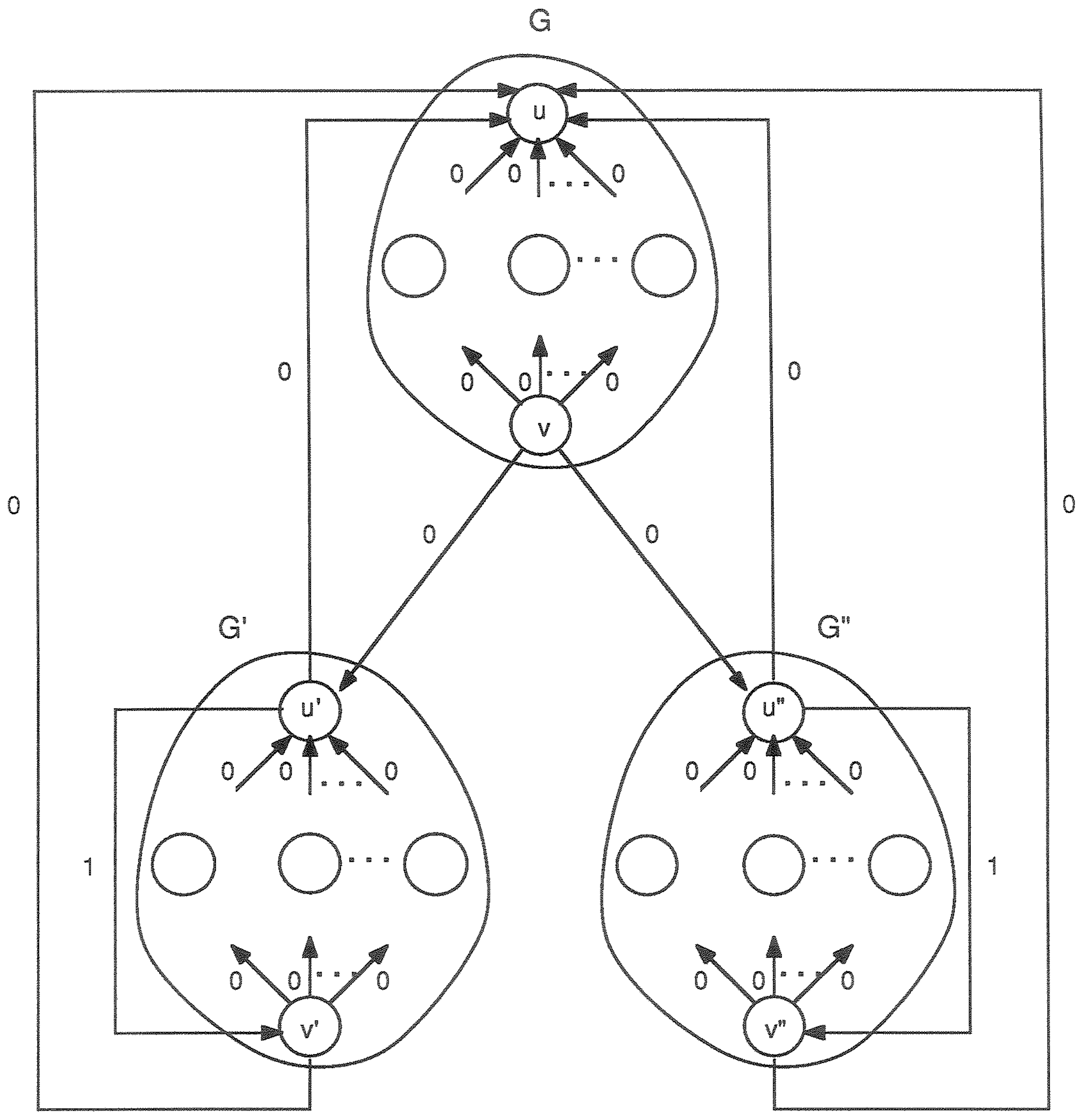


Figure 4.4: Eccentricity (radius) of all nodes is 1 iff <G,u,v>∈ GAP and <G',u',v'>∈ co-GAP.

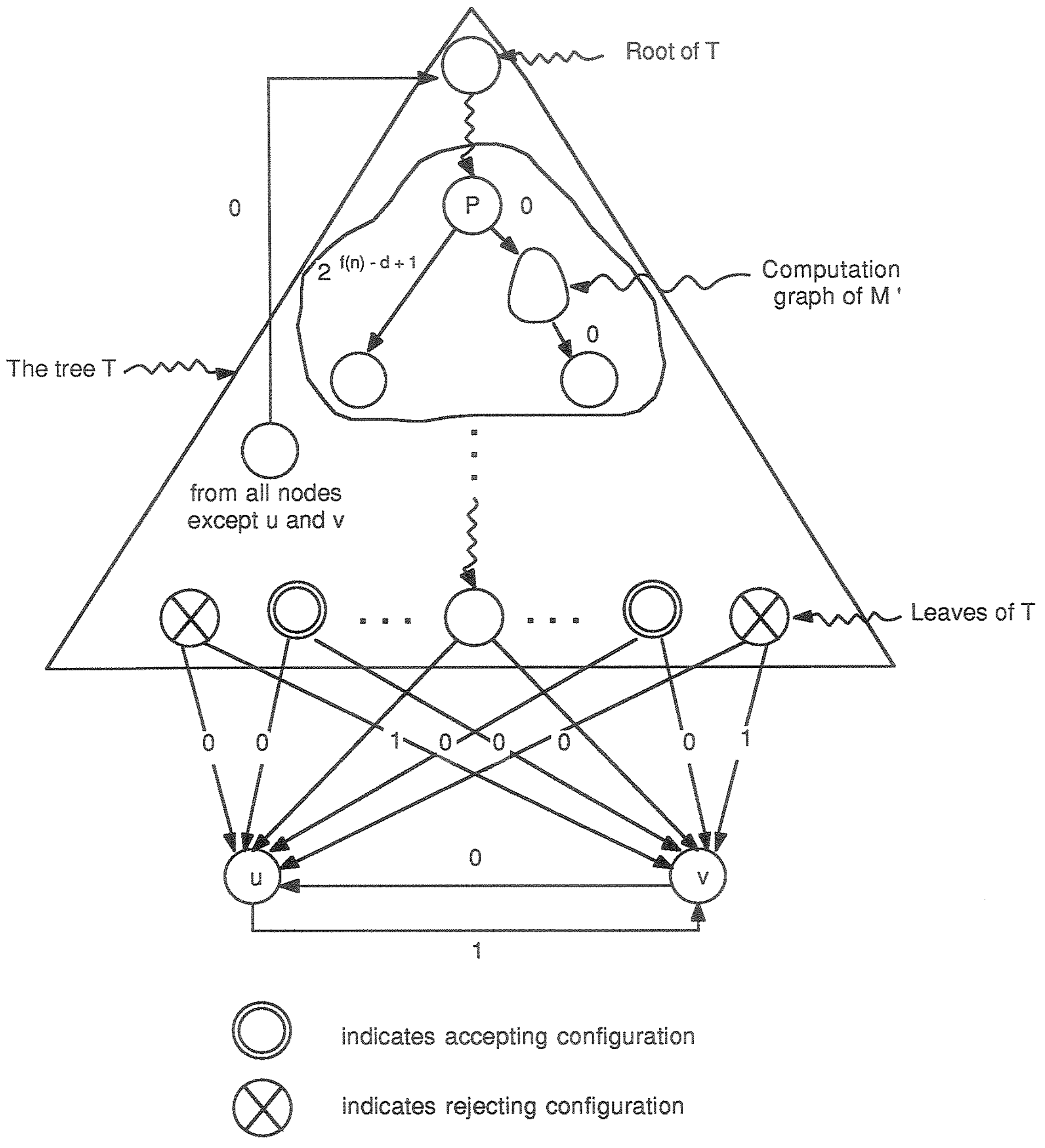

Root of T
0
P
0
$2^{f(n) - d + 1}$
Computation graph of M '
0
The tree T
from all nodes except u and v
Leaves of T
0
0
1
0
0
0
0
1
0
u
v
1
indicates accepting configuration
indicates rejecting configuration

Figure 4.5: Reduction to center of a graph.