Chicago Journal of Theoretical Computer Science

MIT Press

Volume 1995, Article 1
30 June, 1995

ISSN 1073–0486. MIT Press Journals, 55 Hayward St., Cambridge, MA 02142; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://www-mitpress.mit.edu/jrnls-catalog/chicago.html/*
- *http://www.cs.uchicago.edu/publications/cjtcs/*
- *gopher.mit.edu*
- *gopher.cs.uchicago.edu*
- anonymous *ftp* at *mitpress.mit.edu*
- anonymous *ftp* at *cs.uchicago.edu*

©1995 by Massachusetts Institute of Technology. Subscribers are licensed to use journal articles in a variety of ways, limited only as required to insure fair attribution to authors and the journal, and to prohibit use in a competing commercial product. See the journal's World Wide Web site for further details. Address inquiries to the Subsidiary Rights Manager, MIT Press Journals; (617)253-2864; *journals-rights@mit.edu*.

The *Chicago Journal of Theoretical Computer Science* is a peer-reviewed scholarly journal in theoretical computer science. The journal is committed to providing a forum for significant results on theoretical aspects of all topics in Computer Science.

Editor in chief: Janos Simon

Consulting editors: Joseph Halpern, Stuart A. Kurtz, Raimund Seidel

Editors: Martin Abadi, Pankaj Agarwal, Eric Allender, Tetsuo Asano, Laszló Babai, Eric Bach, Stephen Brookes, Jin-Yi Cai, Anne Condon, Cynthia Dwork, David Eppstein, Ronald Fagin, Lance Fortnow, Steven Fortune, Greg Frederickson, Andrew Goldberg, Georg Gottlob, Vassos Hadzilacos, Juris Hartmanis, Maurice Herlihy, Stephen Homer, Neil Immerman, Paris Kanellakis, Howard Karloff, Philip Klein, Phokion Kolaitis, Stephen Mahaney, Michael Merritt, John Mitchell, Ketan Mulmuley, Gil Neiger, David Peleg, Andrew Pitts, James Royer, Alan Selman, Nir Shavit, Eva Tardos, Sam Toueg, Moshe Vardi, Jennifer Welch, Pierre Wolper

Managing editor: Michael J. O'Donnell

Electronic mail: *chicago-journal@cs.uchicago.edu*

Symmetric *Logspace* is Closed Under Complement

Noam Nisan Amnon Ta-Shma

30 June, 1995

Abstract

Abstract-1 We present a *Logspace*, many-one reduction from the undirected s–t connectivity problem to its complement. This shows that $SL = coSL$.

1 Introduction

1-1 This paper deals with the complexity class symmetric *Logspace*, *SL*, defined by Lewis and Papadimitriou in [LP82]. This class can be defined in several equivalent ways:

1. Languages that can be recognized by symmetric, nondeterministic Turing Machines that run within logarithmic space. See [LP82].
2. Languages that can be accepted by a uniform family of polynomial-size contact schemes (these may also be called switching networks). See [Raz91].
3. Languages that can be reduced in *Logspace* via a many-one reduction to *USTCON*, the undirected s–t connectivity problem.

1-2 A major reason for the interest in this class is that it captures the complexity of *USTCON*. The input to *USTCON* is an undirected graph G, and two vertices in it, s and t. The input should be accepted if s and t are connected via a path in G. The similar problem *STCON*, where the graph G has directed edges, is complete for non-deterministic *Logspace* (*NL*). Several combinatorial problems are known to be in *SL* or *coSL* (e.g., 2-colorability is complete for *coSL* [Rei82]).

1-3 The following facts are known regarding *SL* relative to other complexity classes in "the vicinity":

$$L \subseteq SL \subseteq RL \subseteq NL$$

Here, L is the class deterministic *Logspace* and *RL* is the class of problems that can be accepted with one-sided error by a randomized *Logspace* machine running in polynomial time. The containment $SL \subseteq RL$ is the only non-trivial one in the line above and follows directly from the randomized *Logspace* algorithm for *USTCON* of [AKL+79]. It is also known that $SL \subseteq SC$ [Nis92], $SL \subseteq \bigoplus L$ [KW93], and $SL \subseteq DSPACE(\log^{1.5} n)$ [NSW92].

1-4 After the surprising proofs that *NL* is closed under complement were found [Imm88, Sze88], Borodin et al. [BCD+89] asked whether the same is true for *SL*. They could prove only the weaker statement, namely that $SL \subseteq coRL$, and left "$SL = coSL$?" as an open problem. In this paper we solve the problem in the affirmative by exhibiting a *Logspace*, many-one reduction from *USTCON* to its complement. Quite surprisingly, the proof of our theorem does not use inductive counting, as do the proofs of $NL = coNL$. This is a simpler proof than Borodin's, et al.; however, it uses the [AKS83] sorting networks.

Theorem 1 $SL = coSL$

It should be noted that the monotone analogs (see [GS91]) of *SL* and *coSL* are known to be different [KW88].

1-5 As a direct corollary of our theorem, we obtain $L^{\langle SL \rangle} = SL$ where $L^{\langle SL \rangle}$ is the class of languages accepted by *Logspace* oracle Turing machines with oracle from *SL*, using the oracle model of [RST84].

Corollary 1.1 $L^{\langle SL \rangle} = SL$

1-6 In particular, we show that both "symmetric *Logspace* hierarchies" (the one defined by alternation in [Rei82] and the one defined by oracle queries in [BPS92]) collapse to *SL*.

2 Proof of Theorem

2.1 Overview of proof

2.1-1 We design a many-one reduction from *coUSTCON* to *USTCON*. We start by developing simple tools for combining reductions in subsection 2.2. In particular, these tools will allow us to use the AKS sorting networks in order to "count." At this point, the main ingredient of the reduction will be the calculation of the number of the connected components of a graph. An upper bound to this number is easily obtained using transitive closure, while the main idea of the proof is to obtain a lower bound by computing a spanning forest of the graph. We do this in subsection 2.3. Everything is put together in subsection 2.4.

2.2 Projections to *USTCON*

2.2-1 In this paper we will use only the simplest kind of reductions (i.e., *Logspace*-uniform projection reductions [SV85]). Moreover, we will be interested only in reductions to *USTCON*. We define this kind of reduction and we show some of its basic properties in this subsection.

Notation 2.1 *Given* $f\colon\{0,1\}^* \to \{0,1\}^*$, *denote by* $f_n\colon\{0,1\}^n \to \{0,1\}^*$ *the restriction of* f *to inputs of length* n. *Denote by* $f_{n,k}$ *the* k*th bit function of* f_n *(i.e., if* $f_n\colon\{0,1\}^n \to \{0,1\}^m$, *then* $f_n(\vec{x}) = (f_{n,1}(\vec{x}), \ldots, f_{n,m}(\vec{x}))$*).*

Notation 2.2 *We represent an* n*-node undirected graph* G *using* $\binom{n}{2}$ *variables* $\vec{x} = (x_{i,j})_{1\le i<j\le n}$ *s.t.* $x_{i,j}$ *is* 1 *iff* $(i,j) \in E(G)$. *If* $f(\vec{x})$ *operates on graphs, we will write* $f(G)$, *meaning that the input to* f *is a binary vector of length* $\binom{n}{2}$ *representing* G.

2.2-2 We say that $f\colon\{0,1\}^* \to \{0,1\}^*$ reduces to *USTCON*(m) if we can, uniformly and in *Logspace*, label the edges of a graph of size m with $\{\, 0, 1, x_i, \neg x_i \mid 1 \le i \le n \,\}$, s.t. $f_{n,k}(\vec{x}) = 1$ if and only if there is a path from 1 to k in the corresponding graph. We may formalize this with a definition.

Definition 2.1 *We say that* $f\colon\{0,1\}^* \to \{0,1\}^*$ *reduces to USTCON*(m), $m = m(n)$, *if there is a uniform family of Space*$(log(n))$ *functions* $\{\sigma_{n,k}\}$ *s.t. for all* n *and* k*:*

- $\sigma_{n,k}$ *is a projection (i.e.,* $\sigma_{n,k}$ *is a mapping from* $\{\,\{i,j\} \mid 1 \le i < j \le m\,\}$ *to* $\{\,0, 1, x_i, \neg x_i \mid 1 \le i \le n\,\}$*). We abuse the notation by writing* $\sigma_{n,k}(i,j)$ *instead of* $\sigma_{n,k}(\{i,j\})$*.*
- *Given* $\vec{x}$ *define* $G_{\vec{x},k}$ *to be the graph* $G_{\vec{x},k} = (\{1,\ldots,m\}, E)$ *where*

$$E = \{\,(i,j) \mid \sigma_{n,k}(i,j) = 1 \ \ or \ \ (\sigma_{n,k}(i,j) = x_i \ \ and \ \ x_i = 1) \\ or \ \ (\sigma_{n,k}(i,j) = \neg x_i \ \ and \ \ x_i = 0)\,\}$$

- $f_{n,k}(\vec{x}) = 1 \iff$ *there is a path from* 1 *to* m *in* $G_{\vec{x},k}$*.*

If σ *is restricted to the set* $\{\,0, 1, x_i \mid 1 \le i \le n\,\}$*, we say that* f monotonically *reduces to USTCON*(m)*.*

Lemma 2.1 *If* f *has uniform monotone formulae of size* $s(n)$*, then* f *is monotonically reducible to USTCON*$(O(s(n)))$*.*

Proof of Lemma 2.1 Given a formula ϕ, recursively build (G, s, t) as follows:

- If $\phi = x_i$, then build a graph with two vertices s and t and one edge between them labeled x_i.
- If $\phi = \phi_1 \wedge \phi_2$, and (G_i, s_i, t_i), then the graphs for ϕ_i, $i = 1, 2$ identify s_2 with t_1, and define $s = s_1, t = t_2$.
- If $\phi = \phi_1 \vee \phi_2$, and (G_i, s_i, t_i), then the graphs for ϕ_i, $i = 1, 2$ identify s_1 with t_1 and s_2 with t_2, and define $s = s_1 = t_1$ and $t = s_2 = t_2$.

Proof of Lemma 2.1 □

2.2-3

Definition 2.2 *Sort*: $\{0,1\}^n \rightarrow \{0,1\}^n$ *is the Boolean sorting function (i.e., it moves all the zeros to the beginning of the string).*

Using the AKS sorting networks [AKS83], which belong to NC^1, we derive the following corollary.

Corollary 2.2 *Sort is monotonically reducible to USTCON*$(poly)$*.*

Lemma 2.3 *If f monotonically reduces to USTCON(m_1), and g reduces to USTCON(m_2), then $f \circ g$ reduces to USTCON$(m_1^2 \cdot m_2)$, where $\circ$ is the standard function composition operator.*

Proof of Lemma 2.3 The function f monotonically reduces to a graph with m_1 vertices, where each edge is labeled with one of $\{0, 1, x_i\}$. In the composition function $f \circ g$, each x_i is replaced by $x_i = g_i(\vec{y})$, which can be reduced to a connectivity problem of size m_2. Replace each edge labeled x_i with its corresponding connectivity problem. There can be m_1^2 edges, each replaced by a graph with m_2 vertices. The resulting new graph has $m_1^2 \cdot m_2$ vertices.

Proof of Lemma 2.3 □

2.3 Finding a spanning forest

2.3-1 We show how to build a spanning forest using *USTCON* in this section. Reif [Rei82], and independently, Cook, have previously noted this idea.

2.3-2 Given a graph G, index the edges from 1 to m. We can view the indices as weights to the edges, and, as no two edges have the same weight, we know that there is a unique minimal spanning forest F. In our case, where the edges are indexed, this minimal forest is the lexicographically first spanning forest.

2.3-3 It is well known that the greedy algorithm finds a minimal spanning forest. Let us recall how the greedy algorithm works in our case. The algorithm builds a spanning forest F which is empty at the beginning, $F = \emptyset$. Then the algorithm checks the edges one by one according to their order. For each edge e, if e does not close a cycle in F, then e is added to the forest. This may be stated as $F = F \cup \{e\}$.

2.3-4 At first glance, the algorithm looks sequential. However, claim 2.4.1 shows that the greedy algorithm is actually highly parallel. To check whether an edge participates in the forest, we need only one s–t connectivity problem over an easily-obtainable graph.

Definition 2.3 *For an undirected graph G denote by $LFF(G)$ the lexicographically first spanning forest of G. Let $SF(G) \in \{0,1\}^{\binom{n}{2}}$ be:*

$$SF_{i,j}(G) = \begin{cases} 0 & (i,j) \in LFF(G) \\ 1 & \textit{otherwise} \end{cases}$$

Lemma 2.4 *SF reduces to USTCON(poly).*

Proof of Lemma 2.4 Let F be the lexicographically first spanning forest of G. For $e \in E$, define G_e to be the subgraph of G containing only the edges $\{\, e' \in E \mid index(e') < index(e) \,\}$.

Claim 2.4.1 $e = (i, j) \in F \iff e \in E$ *and* i *is not connected to* j *in* G_e.

Proof of Claim 2.4.1 Let $e = (i, j) \in E$. Denote by F_e the forest which the greedy algorithm built at the time it was checking e. So $e \in F$ if and only if e does not close a cycle in F_e.

($\Longrightarrow$) $e \in F$ and therefore, e does not close a cycle in F_e, but then e does not close a cycle in the transitive closure of F_e, and in particular e does not close a cycle in G_e.

($\Longleftarrow$) e does not close a cycle in G_e therefore, e does not close a cycle in F_e and $e \in F$.

Proof of Claim 2.4.1 □

Therefore, $SF_{i,j}(G) = \neg x_{i,j}$ or i is connected to j in $G_{(i,j)}$. Because $\neg x_{i,j}$ can be viewed as the connectivity problem over the graph with two vertices and one edge labeled $\neg x_{i,j}$, it follows from lemmas 2.1 and 2.3 that *SF* reduces to *USTCON*. Notice, however, that the reduction is not monotone.

Proof of Lemma 2.4 □

2.4 Putting it together

2.4-1 First, we want to build a function that takes one representative from each connected component. We define $LI_i(G)$ to be 0 if and only if the vertex i has the largest index in its connected component.

Definition 2.4 $LI(G) \in \{0, 1\}^n$

$$LI_i(G) = \begin{cases} 0 & i \text{ has the largest index in its connected component} \\ 1 & \text{otherwise} \end{cases}$$

Lemma 2.5 *LI reduces to USTCON(poly).*

Proof of Lemma 2.5 $LI_i(G) = \bigvee_{j=i+1}^{n}(i \text{ is connected to } j \text{ in } G)$. So LI is a simple monotone formula over connectivity problems, and LI reduces to *USTCON* by lemmas 2.1 and 2.3. This is, actually, a monotone reduction.

Proof of Lemma 2.5 □

2.4-2 Using the spanning forest and the LI function, we can exactly compute the number of connected components of G (i.e., given G, we can compute a function NCC_i which is 1 iff there are exactly i connected components in G).

Definition 2.5 $NCC(G) \in \{0,1\}^n$

$$NCC_i(G) = \begin{cases} 1 & \textit{there are exactly } i \textit{ connected components in } G \\ 0 & \textit{otherwise} \end{cases}$$

Lemma 2.6 *NCC reduces to USTCON(poly).*

Proof of Lemma 2.6 Let F be a spanning forest of G. It is easy to see that if G has k connected components, then $|F| = n - k$. Define:

$$\begin{aligned} f(G) &= Sort \circ LI(G) \\ g(G) &= Sort \circ SF(G) \end{aligned}$$

Then:

$$\begin{aligned} f_i(G) = 1 &\implies k < i \\ g_i(G) = 1 &\implies n - k < i \implies k > n - i, \end{aligned}$$

and thus:

$$NCC_i(G) = f_{i+1}(G) \wedge g_{n-i+1}(G)$$

Therefore, applying lemmas 2.1, 2.2, 2.3, 2.4, and 2.5 proves the lemma.

Proof of Lemma 2.6 □

2.4-3 Finally, we can reduce the non-connectivity problem to the connectivity problem, thus proving that $SL = coSL$.

Lemma 2.7 *coUSTCON reduces to USTCON(poly).*

Proof of Lemma 2.7 Given (G, s, t), define G^+ to be the graph $G \cup \{(s, t)\}$. Denote by $\#CC(H)$ the number of connected components in the undirected graph H.

$$s \text{ is not connected to } t \text{ in } G \iff \#CC(G^+) = \#CC(G) - 1$$
$$\iff \bigvee_{i=2}^{n} NCC_i(G) \wedge NCC_{i-1}(G^+)$$

Therefore, applying lemmas 2.1, 2.3, and 2.6 proves the lemma.

Proof of Lemma 2.7 □

3 Extensions

3-1 Denote by $L^{\langle SL \rangle}$ the class of languages accepted by *Logspace* oracle Turing machines with an oracle from *SL*. An oracle Turing machine has a work tape and a write-only query tape (with unlimited length) which is initialized after every query. We get:

Corollary 3.1 $L^{\langle SL \rangle} = SL$.

Corollary 3.1 Proof-1 **Proof of Corollary 3.1** Let M be an oracle Turing Machine running in $L^{\langle SL \rangle}$, and fix an input $\vec{x}$ to M. We build the "configuration" graph $G(V, E)$ of M using the following process:

- Let V contain all possible configurations.
- Then, $(v, w) \in E$ with the label "q is (not) s–t connected," if, starting from configuration v, the next query is q. If the oracle answers that "q is (not) connected," then the machine moves to configuration w.

Corollary 3.1 Proof-2 Notice that we can ignore the direction of the edges, as backward edges do not benefit us. The reason is that from any vertex v there is only one forward edge leaving v that can be traversed (i.e., whose label matches the oracle's answer). Therefore, if we reach v using a "backward edge" $w \to v$, then the only forward edge leaving v that can be traversed is $v \to w$.

Corollary 3.1 Proof-3 Now we can replace query edges labeled "q is connected" with the s–t connectivity problem q, and edges labeled "q is not connected" with the s–t connectivity problem obtained using our theorem that $SL = coSL$, resulting in one, not too big, s–t connectivity problem. It is also clear that this can be done in *Logspace*, completing the proof.

Proof of Corollary 3.1 □

3-2 As the symmetric *Logspace* hierarchy defined in [Rei82] is known to be within $L^{\langle SL \rangle}$, this hierarchy collapses to SL.

3-3 As can be easily seen, the above argument holds for any undirected graph with undirected query edges, which is exactly the definition of $SL^{\langle SL \rangle}$ given by [BPS92]. Thus, $SL^{\langle SL \rangle} = SL$, and, by induction, the SL hierarchy defined in [BPS92] collapses to SL.

4 Acknowledgments

4-1 We would like to thank Amos Beimel, Allan Borodin, Assaf Schuster, Robert Szelepcsényi, and Avi Wigderson for helpful discussions.

Acknowledgement of support: This work was supported by BSF Grant 92-00043 and by a Wolfeson Award administered by the Israeli Academy of Sciences. The work was revised while visiting BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation. A preliminary version of this paper appeared in the proceedings of the *27th Annual ACM Symposium on the Theory of Computing, 1995*.

References for CJTCS Volume 1995, Article 1

References

[AKL+79] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal sequences and the complexity of maze problems. In *Proceedings of the 20th Annual IEEE Symposium on the Foundations of Computer Science*. Institute of Electrical and Electronics Engineers, 1979.

[AKS83] M. Ajtai, J. Komlos, and E. Szemeredi. An $O((n \log n))$ sorting network. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 1–9. Association for Computing Machinery, 1983.

[BCD+89] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, 1989.

[BPS92] Y. Ben-Asher, D. Peleg, and A. Schuster. The complexity of reconfiguring networks models. In *Proceedings of the Israel Symposium on the Theory of Computing and Systems*, May 1992. To appear in *Information and Computation*.

[GS91] M. Grigni and M. Sipser. Monotone separation of logspace from NC^1. In *Annual Conference on Structure in Complexity Theory*, 1991.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17, 1988.

[KW88] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 539–550. Association for Computing Machinery, 1988.

[KW93] M. Karchmer and A. Wigderson. On span programs. In *Annual Conference on Structure in Complexity Theory*, 1993.

[LP82] H. R. Lewis and C. H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19, 1982.

[Nis92] N. Nisan. $RL \subseteq SC$. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 619–623. Association for Computing Machinery, 1992.

[NSW92] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O((\log^{1.5} n))$ space. In *Proceedings of the 33th IEEE Symposium on Foundations of Computer Science*, pages 24–29. Institute of Electrical and Electronics Engineers, 1992.

[Raz91] A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. In *Fundamentals of Computation Theory: 8th International Conference, Lecture Notes in Computer Science,* 529, pages 47–60, New York/Berlin, 1991. Springer-Verlag.

[Rei82] J. H. Reif. Symmetric complementation. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 201–214. ACM SIGACT, 1982.

[RST84] L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28:216–230, April 1984.

[SV85] S. Skyum and L. Valiant. A complexity theory based on boolean algebra. *Journal of the ACM*, 1985.

[Sze88] R. Szelepcsenyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26, 1988.