RECURRENCE-BASED HEURISTICS FOR THE HAMILTONIAN PATH INCLUSION AND EXCLUSION ALGORITHM

ERIC BAX*

Abstract. For many problem instances, the inclusion and exclusion formula has many cancellations and symmetries. By imposing a hierarchy on the formula's terms, we develop general reductions for inclusion and exclusion algorithms. We apply these reductions to an algorithm which counts Hamiltonian paths, and we develop a branch and bound algorithm to detect Hamiltonian paths. Then we show how to apply the reductions to other inclusion and exclusion algorithms.

Key words. algorithms, combinatorial problems, parallel algorithms.

AMS subject classifications. 05,68

1. Introduction. The principle of inclusion and exclusion is the basis for several combinatorial algorithms, including counting graph colorings [12]; computing the matrix permanent [10]; counting Hamiltonian paths, sequencing, and bin packing [6]; counting satisfying assignments to a DNF formula [7]; and counting paths and cycles [2]. These algorithms have exponential time complexity, as do their dynamic programming counterparts (e.g. see [5, 3].) But the inclusion and exclusion algorithms have polynomial space requirements, while their dynamic programming counterparts require exponential space. Also, the inclusion and exclusion algorithms can be efficiently computed using message-passing parallel machines because the lack of data interdependencies among computed terms means that very little communication is required [1]. Our top-down approach to reduction development ensures that these algorithms remain efficient on parallel machines.

2. Inclusion and Exclusion. Inclusion and exclusion counts the members of a universal set U that do not belong to any of the sets $P_1 \ldots P_n$.

DEFINITION 2.1. $N(S) \equiv |\bigcap_{j \in S} P_j| \; \forall S \neq \emptyset$, *and* $N(\emptyset) \equiv |U|$. $N(S)$ *counts the members of* U *which belong to every set indexed by* S.

The inclusion and exclusion formula is:

$$|U - (P_1 \cup \ldots \cup P_n)| = \sum_{S \subseteq \{1,\ldots,n\}} (-1)^{|S|} N(S) \tag{1}$$

3. Recursion. DEFINITION 3.1. $N(R,S) \equiv |(\bigcap_{i \in R} \overline{P_i}) \cap (\bigcap_{j \in S} P_j)|$. $N(R,S)$ *counts the members of* U *which are not in any set indexed by* R *and which are in every set indexed by* S.

THEOREM 3.2 (RECURSION).

$$N(\emptyset, S) = N(S) \tag{2}$$

$$N(R,S) = N(R - \{r\}, S) - N(R - \{r\}, S \cup \{r\}) \; \forall r \in R \tag{3}$$

* Supported by an NSF fellowship. Computer Science Department, California Institute of Technology 256-80, Pasadena, California, 91125 (eric@csvax.caltech.edu).

Proof. The base case (2) follows directly from the definitions of $N(S)$ and $N(R,S)$. For the recursion (3), let $\hat{U} \equiv (\bigcap_{i\in R-\{r\}} \overline{P_i}) \cap (\bigcap_{j\in S} P_j)$. Then we can rewrite (3):

$$|\hat{U} \cap \overline{P_r}| = |\hat{U}| - |\hat{U} \cap P_r| \square \tag{4}$$

□

We will use the recursion to compute $N(\{1\dots n\},\emptyset)$, which is equal to $|U - (P_1 \cup \dots \cup P_n)|$ by DeMorgan's Law.

THEOREM 3.3 (HIERARCHY).

$$N(R,S) = \sum_{A\subseteq R} (-1)^{|A|} N(S \cup A) \tag{5}$$

Proof: Apply recursion (2) and (3) to $N(R,S)$ $|R|$ times. □

Note that applying (5) to $N(\{1\dots n\},\emptyset)$ gives the inclusion and exclusion formula.

4. Cancellation Theorems. THEOREM 4.1. $N(R,S) \geq 0$.

Proof: N(R,S) is the size of a set.

THEOREM 4.2. *$\forall\ Q \subseteq R$ and $T \subseteq S$, $N(Q,T) \geq N(R,S)$.*

Proof: The set counted by $N(R,S)$ is a subset of the set counted by $N(Q,T)$.

THEOREM 4.3. *If $\exists M \subseteq R$ such that $\forall\ A,B \subseteq M$ with $|A| = |B|$, $N(R-M,S\cup A) = N(R-M,S\cup B)$ then:*

$$N(R,S) = \sum_{i=0}^{|M|} (-1)^i C(|M|,i) N(R-M,S\cup A_i) \tag{6}$$

where A_i is any size i subset of M.

Proof: Expand $N(R,S)$ by $|M|$ recursions (3), using the elements of M as the "split" elements. Collect the resulting terms $N(R-M,\hat{S})$ according to $|\hat{S}|$.

5. Reductions.

5.1. R Subset Reduction. For $|T| = |S|$, Theorem 4.2 supplies bounds for $N(R,S)$ which increase in accuracy and computational expense as $|Q|$ increases. $|Q| = 0$ yields $N(\emptyset,S) \geq N(R,S)$. $|Q| = 1$ yields $N(\{r\},S) = N(\emptyset,S) - N(\emptyset,S\cup\{r\}) \geq N(R,S)$ for every $r \in R$.

By Theorems 4.1 and 4.2, $N(Q,S) \geq N(R,S) \geq 0$, so $N(Q,S) = 0$ implies $N(R,S) = 0$. In this case $N(R,S)$ need not be computed by (3), so $O(2^{|R|} poly(n))$ time is saved.

5.2. Symmetry Reduction. If we can find a set M meeting the conditions of Theorem 4.3, then we save $O(2^{|R|} - (|M|+1)2^{|R-M|})$ computation time by using (6) instead of (3) to compute $N(R,S)$. If $|M| = 1$ then (6) reduces to (3).

5.3. S Subset Reduction. By Theorem 4.2, in (6), $N(R-M,S\cup A_i) \geq N(R-M,S\cup A_{i+1})$. So, in computing (6), if $N(R-M,S\cup A_i) = 0$ for some i, then the remaining terms are all zero. Using this information saves $O((|M|-i)2^{|R-M|})$ computation time.

6. Counting Hamiltonian Paths.

6.1. Definitions and Notation. Given a graph G, with $V = \{s,t\} \cup \{1,\ldots,n\}$, to count s-t Hamiltonian paths, let U be all length $n+1$ s-t walks in G, and let P_i be the walks that do not pass through vertex i. Then $N(R,S)$ is the number of length $n+1$ s-t walks which contain all vertices in R and no vertices in S. So $N(\{1\ldots n\},\emptyset)$ is the number of s-t Hamiltonian paths in G.

Let $G(S)$ be the subgraph induced by $V-S$. Then $N(R,S)$ is the number of length $n+1$ s-t walks in $G(S)$ that contain every vertex in R, and $N(S)$ is the number of length $n+1$ s-t walks in $G(S)$. We can compute $N(S)$ in $poly(n)$ time by dynamic programming [6] or adjacency matrix multiplication [1].

6.2. Applying the Symmetry Reduction. To satisfy the conditions of Theorem 4.3, it is sufficient to find $M \subseteq R$ such that $\forall\ A,B \subseteq M$ with $|A|=|B|$, $G(S\cup A)$ is isomorphic to $G(S\cup B)$ under a vertex mapping that takes $R-M$ to $R-M$. Testing these conditions for all $M \subseteq R$ is prohibitively expensive, so we impose stronger conditions.

We search for $M \subseteq R$ such that (1) the graph induced by M is either complete or empty, and (2) every $v \in V-(S\cup M)$ is either adjacent to every vertex in M or to no vertex in M. Then $\forall\ A,B \subseteq M$, every isomorphism from $G(S\cup A)$ to $G(S\cup B)$ which maps $M-A$ to $M-B$ and maps each other vertex to itself is adjacency-preserving. Since these isomorphisms trivially map $R-M$ to $R-M$, $N(R-M,S\cup A) = N(R-M,S\cup A)$.

Let E_{ij} be the automorphism on $G(S)$ that swaps vertices i and j and maps each other vertex to itself. We define vertex symmetry function $s(i,j)$ to be true iff E_{ij} is adjacency-preserving. Vertex symmetry is an equivalence relation:

reflexive $s(i,i)$ holds for all i since E_{ii} is the identity automorphism.

symmetric $s(i,j)=s(j,i)$ because $E_{ij}=E_{ji}$.

transitive $s(i,j)$ and $s(j,k)$ implies $s(i,k)$ since compositions of adjacency-preserving E_{ij}'s are adjacency-preserving, and $E_{ik} = E_{ij}\circ E_{jk}\circ E_{ij}$.

The partition of R induced by $s(i,j)$ is composed of vertex sets that meet our conditions for M, and hence satisfy Theorem 4.3. To apply the symmetry reduction, choose M to be the largest set in the partition.

6.3. Algorithm. Augmenting the recursion of Theorem 3.2 with the $|Q|=1$ R subset reduction, the symmetry reduction, and the S subset reduction produces the algorithm:

```
N(R, S)
{
function N(S);
integer i, value, sum = 0;
set A, M;

if (R = ∅) return(N(S)); (* base case *)

for each r ∈ R
    if (N(S) − N(S ∪ {r}) = 0) return(0); (* R subset reduction *)

M=largest set in the partition of R induced by s(); (* symmetry reduction *)

for (i = 0; i ≤ |M|; i + +)
    A=first i elements of M;
    value=C(|M|, i) N(R − M, S ∪ A);
```

if ($value = 0$) return(sum); (* S subset reduction *)
$sum = sum + value$;

return(sum);
}

When $|M| = 1$, i.e. no symmetric vertex set is found, any $r \in R$ can be chosen as the "split" vertex in (3). In practice the $r \in R$ with maximum $N(S \cup \{r\})$ works well for sparse graphs, and the minimum works well for dense graphs.

6.4. Analysis. In the worst case none of the reductions are used, so the computation is a depth first traversal of the binary tree induced by (3), with root $N(\{1 \dots n\}, \emptyset)$, nodes $N(R, S)$, and leaves $N(\emptyset, S)$.

Note that the first node to compute values of $N(S)$ and $N(S \cup \{r\})$ is the ancestor of all nodes that use these values. By keeping a stack of $N(S)$ values computed by ancestors of the current node, $N(S)$ is only computed once for each $S \subseteq \{1 \dots n\}$. Since up to $n + 1$ $N(S)$ values may be computed in a node and the computation tree has depth n, the stack requires only $O(n^2)$ space. So the algorithm presented here maintains the polynomial space requirement of the earlier algorithms.

There are $O(2^n)$ tree nodes, and partitioning R by $s()$ takes $O(n|E|)$ time in each node. Also, computing N(S) requires $O(n|E|)$ time [6] for each $S \subseteq \{1 \dots n\}$. Therefore, the algorithm has time complexity $O(2^n n|E|)$.

6.5. Testing. The algorithm was tested on 100 random directed graphs $G_{n,p}$ (with $V = \{s,t\} \cup \{1 \dots n\}$ and edge probability p) for each $n \in \{4, 6, \dots, 14\}$ and $p \in \{0.00, 0.05, \dots, 1.00\}$. The results for each n follow the pattern shown in Figure 1.

The algorithm's efficiency varies with p. For very low p, $G_{n,p}$ is almost always disconnected, and in this case the R subset reduction finds that there are no s-t Hamiltonian paths in the first $N(R, S)$ call. As p increases, $G_{n,p}$ is more likely to be connected, and the S subset reduction begins to play a role. Then there is a p range in which $G_{n,p}$ transits from almost surely not having an s-t Hamiltonian path to almost surely having one. The effectiveness of the R and S subset reductions decrease during this transition. None of the reductions are very effective in the range of p for which $G_{n,p}$ almost certainly has an s-t Hamiltonian path, but very few of the $(n+1)!$ possible s-t Hamiltonian paths are present. Finally, for p near 1, $G_{n,p}$ is almost certainly very dense, and the symmetry reduction is highly effective.

7. Detection and Bounding Algorithms. The algorithm can be altered to solve the detection problem, i.e. to determine whether or not G contains at least one s-t Hamiltonian path. We follow the branch and bound model [8], but we use the method for counting rather than optimization.

The algorithm begins with a single active node representing $N(\{1 \dots n\}, \emptyset)$. At each step an active node is replaced by an equivalent set of active nodes (branches) or computed $N(S)$ terms (leaves). Together, the values of the active nodes and the computed $N(S)$ terms always sum to $N(\{1 \dots n\}, \emptyset)$. Symmetry is implemented by using a coefficient c to indicate how many duplicate $N(R, S)$ terms each active node represents.

At each step we maintain a lower bound L on the number of s-t Hamiltonian paths:

$$L = \sum_{node(R,S,c) \in active} cL(R,S) + \sum_{leaf(S) \in computed} (-1)^{|S|} N(S) \tag{7}$$

where $L(R,S)$ is a lower bound on the contribution of $N(R,S)$ to the number of Hamiltonian paths.

In successive applications of the recursion (3) to $N(\{1 \ldots n\}, \emptyset)$, $N(R,S)$ terms count positively in the sum if $|S|$ is even and negatively if $|S|$ is odd. Using the bounds given by Theorem 4.1 and Theorem 4.2 with $|Q|$=1, we obtain:

$$L(R,S) = \begin{cases} 0 & \text{if } |S| \text{ is even} \\ -\min_{r \in R} N(S) - N(S \cup \{r\}) & \text{if } |S| \text{ is odd} \end{cases}$$

We also maintain an upper bound U:

$$U = \sum_{node(R,S,c) \in active} cU(R,S) + \sum_{leaf(S) \in computed} (-1)^{|S|} N(,S) \tag{8}$$

where $U(R,S)$ is an upper bound on the contribution of $N(R,S)$ to the number of Hamiltonian paths:

$$U(R,S) = \begin{cases} \min_{r \in R} N(S) - N(S \cup \{r\}) & \text{if } |S| \text{ is even} \\ 0 & \text{if } |S| \text{ is odd} \end{cases}$$

If $L > 0$ then there is an s-t Hamiltonian path in G. If $U = 0$ then there is no s-t Hamiltonian path in G. When $L = U$, the number of s-t Hamiltonian paths has been exactly determined.

The branch and bound detection algorithm is:

```
detect Hamiltonian path()
{
function L(R,S), U(R,S), N∅(S);
integer c, L, U;
set R, S, A, M, active;

L = 0; U := U({1...n}, ∅);
active = {node({1...n}, ∅, 1)};

while (L < U)
    remove some node(R,S,c) from active.
    L := L − cL(R,S); U := U − cU(R,S);
    M:=largest set in the partition of R induced by s();
    if (|M| = |R|) (* Branch to form leaves *)
        for (i = 0; i ≤ |M|; i++)
            A=first i elements of M;
            c=C(|M|, i);
            L := L + (−1)^|S| cN(S ∪ A);
            U := U + (−1)^|S| cN(S ∪ A);
    else (* Branch to form active nodes. *)
        for (i = 0; i ≤ |M|; i++)
            A=first i elements of M;
```

```
            c=C(|M|, i);
            L := L + cL(R − M, S ∪ A);
            U := U + cU(R − M, S ∪ A);
            if (L(R − M, S ∪ A) < U(R − M, S ∪ A))
                insert node(R − M, S ∪ A, c) into active;
    if (L > 0) return(true); (* There is at least one s-t Hamiltonian path. *)
    if (U = 0) return(false); (* There are no s-t Hamiltonian paths. *)
if (L > 0) return(true); else return(false); (* L = U. Exact count accomplished. *)
}
```

In the worst case the detection algorithm mimics the counting algorithm, i.e. every leaf $N(S)$ is computed. Choosing nodes at random from *active* can lead to an exponential number of active nodes. However, always choosing an active node with minimum $|R|$ limits the number of active nodes to $\mathrm{O}(n^2)$.

To observe the algorithm's progress to convergence, return the values of the upper and lower bounds U and L after every branching. To stop as soon as the number of Hamiltonian paths is known to within k, change "while $(L < U)$" to "while $(U - L \leq 2k)$", remove the last three lines of the algorithm, and insert "return $((U + L)/2)$;" at the end of the algorithm.

8. Extension to Other Problems.

8.1. Requirements. To use the counting and detection algorithms for other problems, we must (1) define U and $P_1 \ldots P_n$, (2) have a procedure to compute $N(S)$, and (3) have a procedure to find $M \subseteq R$ that satisfies the conditions of Theorem 4.3. Conditions (1) and (2) are necessary for any inclusion and exclusion algorithm. It is not necessary to implement symmetry, i.e. condition (3) can be trivially satisfied by always taking $M = \{r\}$ for some $r \in R$.

8.2. Permanent. The permanent of a 0-1 square matrix is the number of nonzero terms which have exactly one element from each row and exactly one element from each column. To calculate the permanent, let U be the set of all nonzero terms which have exactly one element from each row, and let P_i be the set of nonzero terms with no element from column i. To calculate $N(S)$, zero the columns indexed by S, and take the product of the row sums [10]. Since the product of row sums is invariant under permutations of the columns, to use the symmetry reduction, make M the largest set of equal columns in R.

8.3. Vertex Coloring. A k-coloring of a graph G, with edge set $E = \{1 \ldots n\}$, is an assignment of k or fewer colorings to the vertices so that no pair of adjacent vertices shares a color. To count k-colorings, let U be the set of all assignments of k or fewer colors to the vertices, and let P_i be the set of assignments in which the vertices joined by edge i share a color. To calculate $N(S)$, note that if a path of edges in S joins a pair of vertices, then they share a color in the assignments counted by $N(S)$. So each component in the graph induced by edge set S is monochrome. Hence, $N(S)$ is k raised to the power of the number of connected components in the graph induced by edge set S [12]. The symmetry situation is similar to that for the Hamiltonian path algorithms. To use the symmetry reduction, make M the edges of the largest clique such that (1) all edges in the clique are in R; and (2) if there is an edge in R, or S, or $\overline{R \cup S}$ connecting any clique vertex to a vertex v, then there are edges in R, or S, or $\overline{R \cup S}$, respectively, connecting each clique vertex to v.

9. Open Questions. There are several areas for further research related to the algorithms presented here. For the R subset reduction we use $|Q|=1$, but it is an open question whether the optimal value is some other constant, or depends on n, $|R|$, or $|S|$. Our search for symmetric vertices is efficient, but it only examines a few of the possible cases. Is there an efficient method to check for more cases of symmetry? Would less efficient methods be appropriate for highly symmetric graphs? Are real problems more likely to be symmetric than random test problems? Our reduction utilizes symmetry among vertices. In highly symmetric graphs it may be worthwhile to examine symmetry among subgraphs as well.

Pósa [9] shows that the minimum p value for which $G_{n,p}$ is almost surely Hamiltonian decreases to zero as n goes to infinity. Shamir [11] shows that the threshold p range in which $G_{n,p}$ is neither almost surely Hamiltonian nor almost surely non-Hamiltonian decreases to zero as n goes to infinity. These results are validated by Figure 6. The counting algorithm's loss of efficiency trails the increasing probability of $G_{n,p}$ having an s-t Hamiltonian path in Figure 1. For large n, is the algorithm efficient in the p threshold range? The algorithm's performance in the p threshold range affects its usefulness as a backup algorithm for probabilistic detection algorithms (e.g. [4]) that work well when $G_{n,p}$ is either almost surely Hamiltonian or almost surely non-Hamiltonian.

Acknowledgements. I thank Dr. Douglas Rall, Dr. P. M. Cook, and Dr. András Recski for introducing me to combinatorial algorithms. I thank Zehra Cataltepe, Paul Sivilotti, Dr. Joel Franklin, and Dr. Yaser Abu-Mostafa for their help and encouragement. Finally, I thank Dr. Hayden Porter for teaching me an important concept – recursion.

REFERENCES

[1] E. Bax, *Inclusion and exclusion algorithm for the Hamiltonian path problem*, Inf. Proc. Lett., 47 (4) (1993), pp. 203-207.

[2] E. Bax, *Counting paths and cycles*, Inf. Proc. Lett., 52 (1994), pp. 249-252.

[3] R. Bellman, *Combinatorial processes and dynamic programming*, in: R. Bellman and M. Hall, Eds., *Combinatorial Analysis*, Proc. AMS Symposia on Applied Mathematics, Vol. X, American Mathematical Society, Providence, RI, 1960.

[4] Y. Gurevich and S. Shelah, *Expected computation time for Hamiltonian path problem*, SIAM J. Comput., 16 (3) (1987), pp. 486-502.

[5] M. Held and R. M. Karp, *A dynamic programming approach to sequencing problems*, J. Soc. Indust. Appl. Math., 10 (1962), pp. 196-210.

[6] R. M. Karp, *Dynamic programming meets the principle of inclusion and exclusion*, Operations Research Letters, 1 (2) (1982), pp. 49-51.

[7] N. Linial and N. Nisan, *Approximate inclusion-exclusion*, Combinatorica, 10 (4) (1990), pp. 349-365.

[8] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization, Algorithms and Complexity* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982, pp. 433-448.

[9] L. Pósa, *Hamiltonian circuits in random graphs*, Discrete Mathematics, 14 (1976), pp. 359-364.

[10] H. J. Ryser, *Combinatorial Mathematics* The Mathematical Association of America, 1963, pp. 24-28.

[11] E. Shamir, *How many random edges make a graph Hamiltonian?* Combinatorica, 3 (1) (1983), pp. 123-131.

[12] H. Whitney, *A logical expansion in mathematics*, Bull. Amer. Math. Soc., 38 (1932), pp. 572-579.

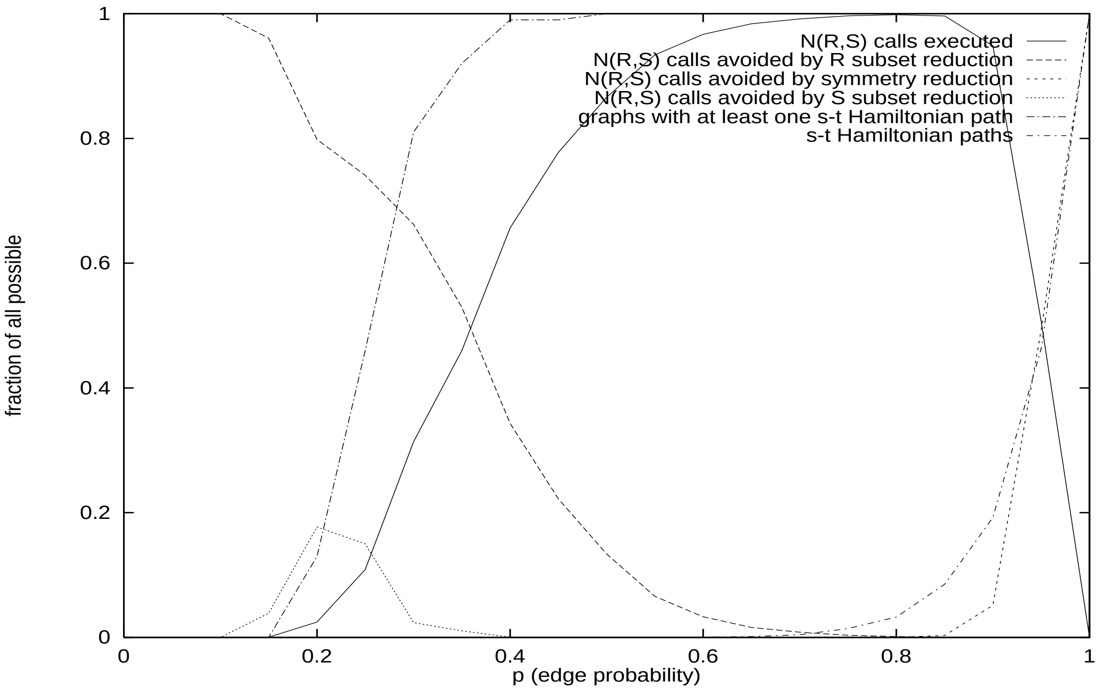

FIG. 1. *Test Results for n=14*

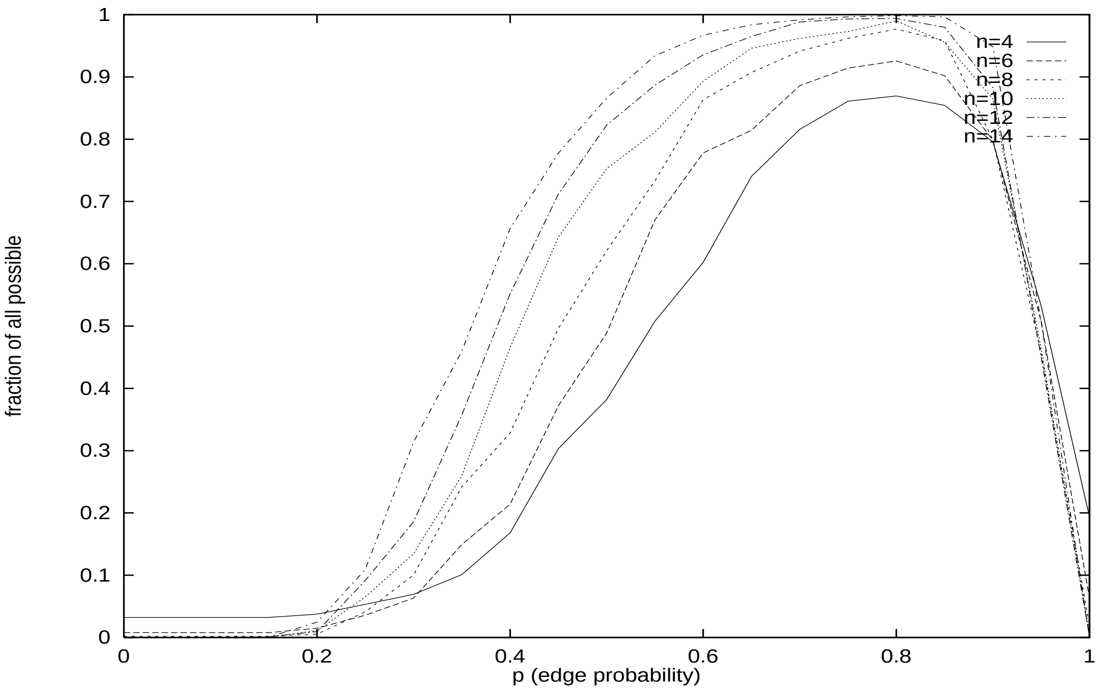

FIG. 2. *N(R,S) Calls Executed*

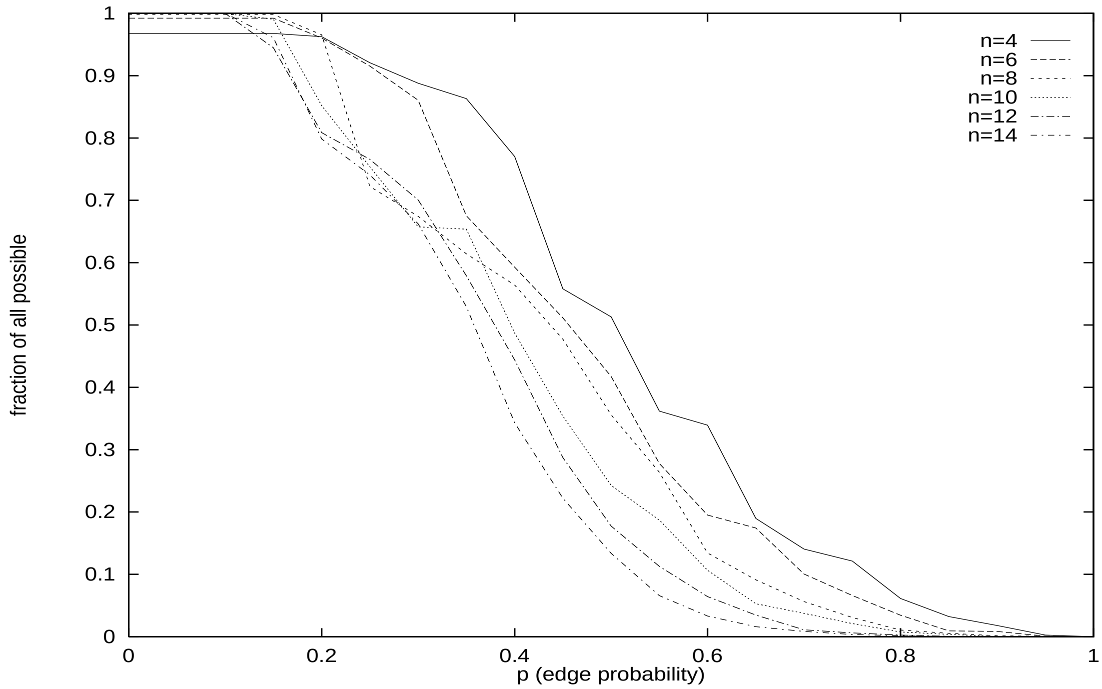

FIG. 3. *N(R,S) Calls Avoided by R Subset Reduction*

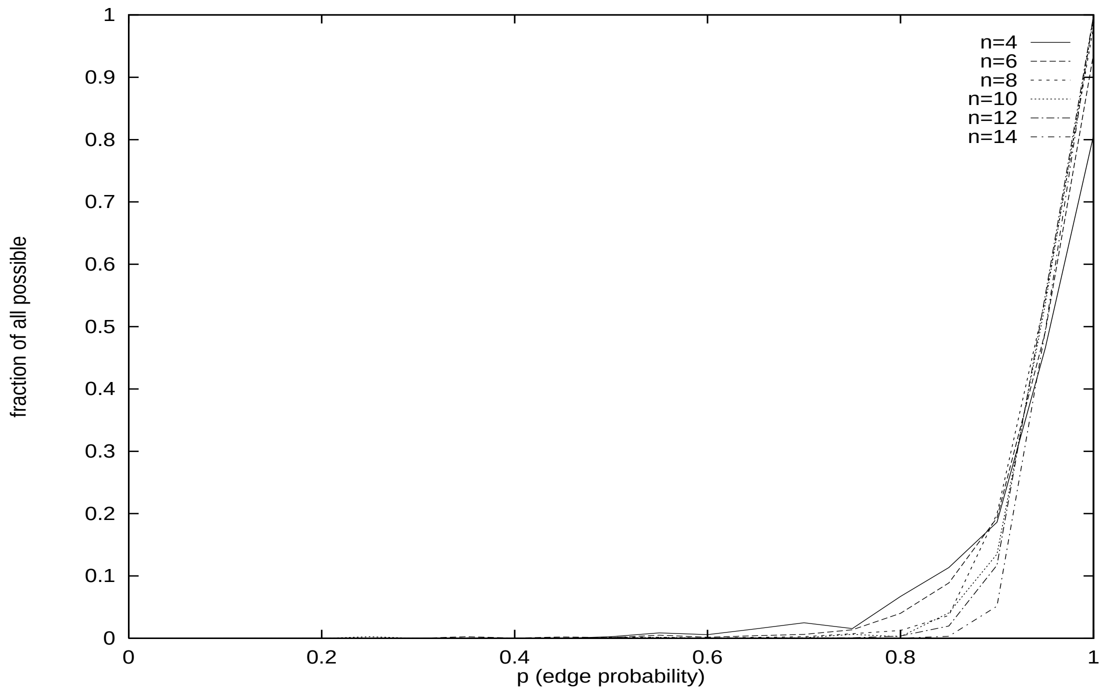

FIG. 4. *N(R,S) Calls Avoided by Symmetry Reduction*

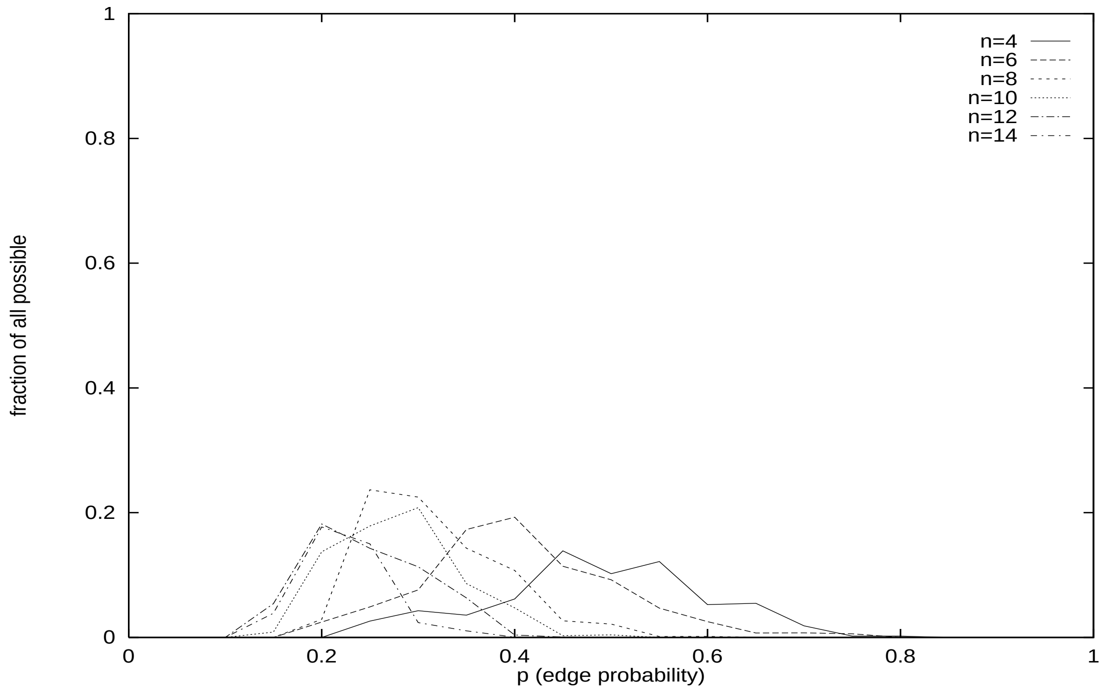

FIG. 5. *N(R,S) Calls Avoided by S Subset Reduction*

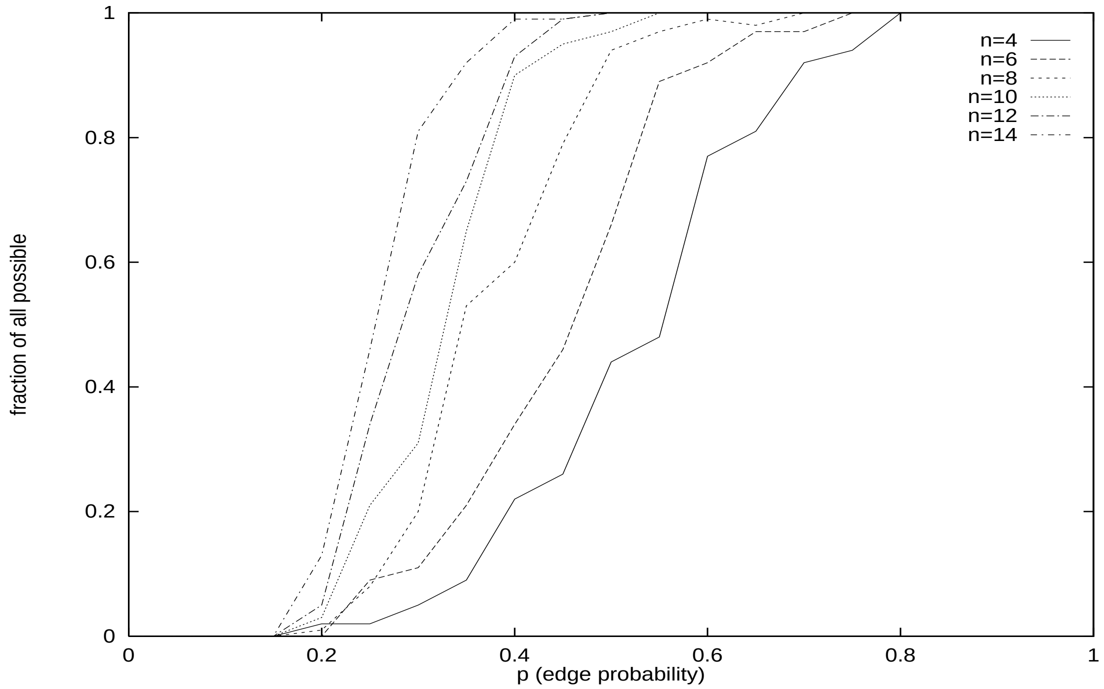

FIG. 6. *Graphs with at Least One Hamiltonian Path*

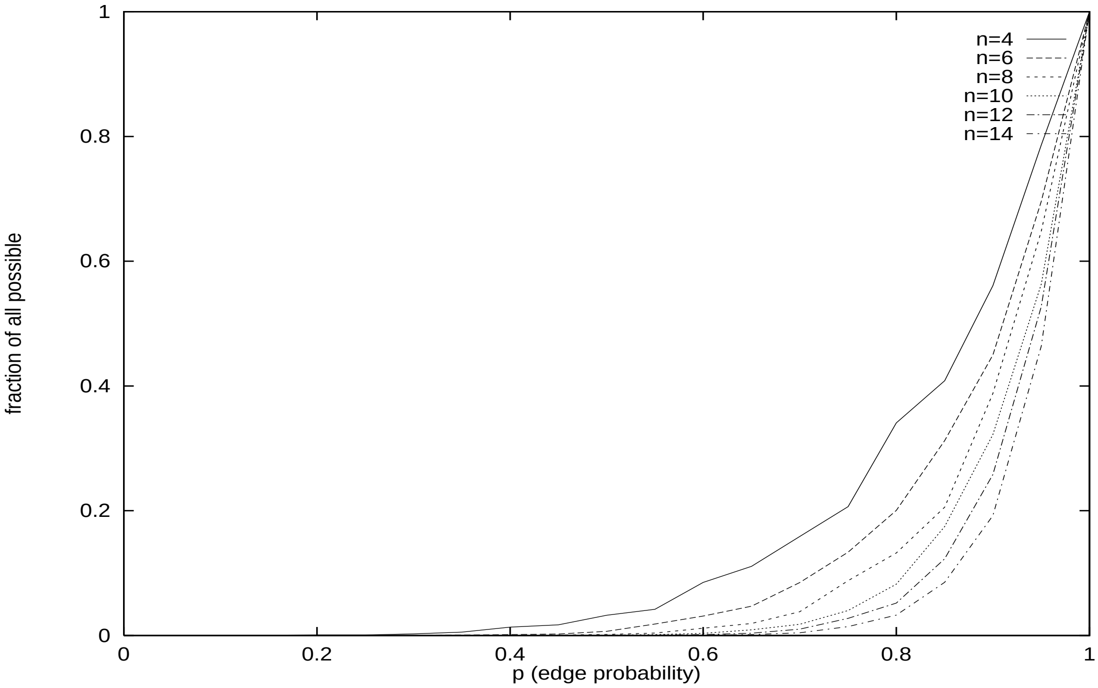

FIG. 7. *s-t Hamiltonian Paths*