Unit 5 - NP AND COMPUTATIONAL INTRACTABILITY

Srinidhi H, Chetan Shetty
Assistant Professor,
Dept of CSE, MSRIT

Topics

- Polynomial-Time Reductions A First Reduction: Independent Set and Vertex Cover.
- Reducing to a More General Case: Vertex Cover to Set Cover.
- NP-Complete Problems: Circuit Satisfiability: A First NP-Complete Problem.
- General Strategy for Proving New Problems NP-Complete.
- Sequencing Problems: The Traveling Salesman Problem, The Hamiltonian Cycle Problem.

1. Polynomial time reductions

Definition: Our basic technique in this exploration is to compare the relative difficulty of different problems; we'd like to formally express statements like, "Problem *X* is at least as hard as problem *Y*." We will formalize this through the notion of *reduction*: we will show that a particular problem *X* is at least as hard as some other problem *Y* by arguing that, if we had a "black box" capable of solving *X*, then we could also solve *Y*. (In other words, *X* is powerful enough to let us solve *Y*.)

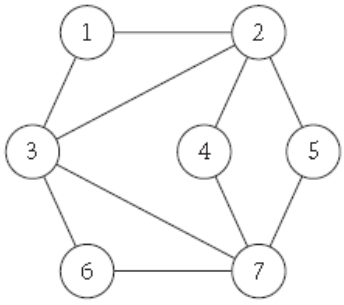
Suppose we had a *black box* that could solve instances of a problem *X*;
If arbitrary instances of problem Y can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to a black box that solves problem X, then we write $Y \leq_P X$; read as "*Y* is polynomial-time reducible to *X*," or "*X* is at least as hard as *Y* (with respect to polynomial time)."

Lemma 1
Suppose $Y \leq_P X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.

Proof: Suppose $Y \leq_P X$ and there actually exists a polynomial-time algorithm to solve X. The algorithm for problem Y that involved a polynomial number of steps plus a polynomial number of calls to the black box. It now becomes an algorithm that involves a polynomial number of steps, plus a polynomial number of calls to a subroutine that runs in polynomial time; in other words, it has become a polynomial-time algorithm.

2. Reduction: Independent Set and Vertex Cover

Optimization version of Independent Set problem:
In a graph *G = (V, E)*, we say a set of nodes $S \subseteq V$ is *independent* if no two nodes in *S* are joined by an edge. It is easy to find small independent sets in a graph (for example, a single node forms an independent set); the hard part is to find a large independent set, since you need to build up a large collection of nodes without ever including two neighbors. For example, the set of nodes {3, 4, 5} is an independent set of size 3 in the graph in Figure, while the set of nodes {1, 4, 5, 6} is a larger independent set.

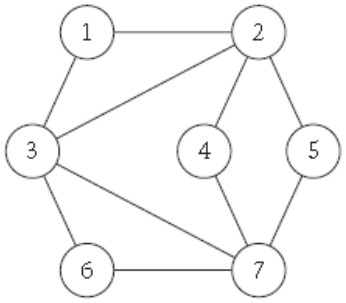


Decision version of Independent set problem:
Given a graph G and a number k, does G contain an independent set of size at least k.

In fact, from the point of view of polynomial-time solvability, there is not a significant difference between the *optimization version* of the problem (find the maximum size of an independent set) and the *decision version* (decide, yes or no, whether G has an independent set of size at least a given k). Given a method to solve the optimization version, we automatically solve the decision version (for any k) as well.

If we can solve the decision version of Independent Set for every k, then we can also find a maximum independent set. For given a graph G on n nodes, we simply solve the decision version of Independent Set for each k; the largest k for which the answer is "yes" is the size of the largest independent set in G. (And using binary search, we need only solve the decision version for $O(\log n)$ different values of k.) This simple equivalence between decision and optimization will also hold in the problems we discuss below.

Vertex Cover. Given a graph $G = (V, E)$, we say that a set of nodes $S \subseteq V$ is a *vertex cover* if every edge $e \in E$ has at least one end in S.



For example, in the graph in Figure above, the set of nodes {1, 2, 6, 7} is a vertex cover of size 4, while the set {2, 3, 7} is a vertex cover of size 3.

3. Reduction:

Lemma 2: *Let $G = (V, E)$ be a graph. Then S is an independent set if and only if its complement $V - S$ is a vertex cover.*

Proof. First, suppose that S is an independent set. Consider an arbitrary edge $e = (u, v)$. Since S is independent, it cannot be the case that both u and v are in S; so one of them must be in $V - S$. It follows that every edge has at least one end in $V - S$, and so $V - S$ is a vertex cover.
Conversely, suppose that $V - S$ is a vertex cover. Consider any two nodes u and v in S. If they were joined by edge e, then neither end of e would lie in $V - S$, contradicting our assumption that $V - S$ is a vertex cover. It follows that no two nodes in S are joined by an edge, and so S is an independent set.

Lemma 3: *Independent Set ≤P Vertex Cover.*

Proof. If we have a black box to solve Vertex Cover, then we can decide whether G has an independent set of size at least k by asking the black box whether G has a vertex cover of size at most $n - k$.

Lemma 4: *Vertex Cover ≤P Independent Set.*

Proof. If we have a black box to solve Independent Set, then we can decide whether G has a vertex cover of size at most k by asking the black box whether G has an independent set of size at least $n - k$.

4. NP Complete Problems

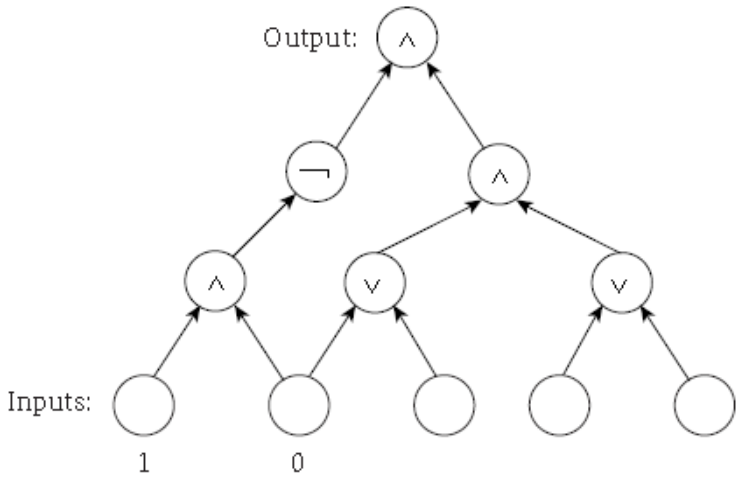
If problem X has following two properties: (i) $X \in$ NP; and (ii) for all $Y \in$ NP, $Y \leq_P X$. In other words, we require that every problem in NP can be reduced to X. We will call such an X an *NP-complete* problem.

***Lemma** :Suppose X is an NP-complete problem. Then X is solvable in polynomial time if and only if P = NP.*

Proof. Clearly, if P = NP, then X can be solved in polynomial time since it belongs to NP. Conversely, suppose that X can be solved in polynomial time. If Y is any other problem in NP, then $Y \leq_P X$ and , it follows that Y can be solved in polynomial time. Hence NP $\subseteq$ P;), we have the desired conclusion.

5. Circuit Satisfiability: A First NP-Complete Problem

Consider the standard Boolean operators that we used to define the Satisfiability Problem: $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT). Our definition of a circuit is designed to represent a physical circuit built out of gates that implement these operators. Thus we define a circuit K to be a labeled, directed acyclic graph such as the one shown in the example of Figure



- The *sources* in K (the nodes with no incoming edges) are labeled either with one of the constants 0 or 1, or with the name of a distinct variable.
- The nodes of the latter type will be referred to as the *inputs* to the circuit.
- Every other node is labeled with one of the Boolean operators $\wedge$, $\vee$, or $\neg$; nodes labeled with $\wedge$ or $\vee$ will have two incoming edges, and nodes labeled with $\neg$ will have one incoming edge.
- There is a single node with no outgoing edges, and it will represent the *output:* the result that is computed by the circuit.

A circuit computes a function of its inputs in the following natural way.

- We imagine the edges as "wires" that carry the 0/1 value at the node they emanate from.
- Each node *v* other than the sources will take the values on its incoming edge(s) and apply the Boolean operator that labels it.
- The result of this ∧, ∨, or ¬ operation will be passed along the edge(s) leaving *v*.
- The overall value computed by the circuit will be the value computed at the output node.

For example, consider the circuit in figure above.

- The leftmost two sources are preassigned the values 1 and 0, and the next three sources constitute the inputs.
- If the inputs are assigned the values 1, 0, 1 from left to right, then we get values 0, 1, 1 for the gates in the second row, values 1, 1 for the gates in the third row, and the value 1 for the output.

Circuit Satisfiability Problem is defined as follows:
"We are given a circuit as input, and we need to decide whether there is an assignment of values to the inputs that causes the output to take the value 1. (If so, we will say that the given circuit is satisfiable, and a satisfying assignment is one that results in an output of 1.)
In our example, we have just seen—via the assignment 1, 0, 1 to the inputs—that the circuit in Figure is satisfiable".

Lemma: Circuit Satisfiability is NP-complete.

Proof: This proof requires that we consider an arbitrary problem *X* in NP, and show that $X \leq P$ Circuit Satisfiability.

We use the fact that any algorithm that takes a fixed number *n* of bits as input and produces a yes/no answer can be represented by a circuit of the type we have just defined: This circuit is equivalent to the algorithm in the sense that its output is 1 on precisely the inputs for which the algorithm outputs yes. Moreover, if the algorithm takes a number of steps that is polynomial in *n*, then the circuit has polynomial size.

We are trying to show that $X \leq P$ Circuit Satisfiability—that is, given an input *s*, we want to decide whether $s \in X$ using a black box that can solve instances of Circuit Satisfiability. Now, all we know about *X* is that it has an efficient certifier $B(\cdot, \cdot)$.

We can use a black box for Circuit Satisfiability as follows.
Since we only care about the answer for a specific input *s*, we view $B(\cdot, \cdot)$ as an algorithm on $n + p(n)$ bits (the input *s* and the certificate *t*), and we convert it to a polynomial-size circuit *K* with $n + p(n)$ sources. The first *n* sources will be hard-coded with the values of the bits in *s*, and the remaining $p(n)$ sources will be labeled with variables representing the bits of *t*; these latter sources will be the inputs to *K*.

Now we simply observe that $s \in X$ if and only if there is a way to set the input bits to *K* so that the circuit produces an output of 1—in other words, if and only if *K* is satisfiable. This establishes that $X \leq P$ *Circuit Satisfiability*, and completes the proof

6. General Strategy for Proving New Problems NP-Complete

Given a new problem X, here is the basic strategy for proving it is NP complete.
1. Prove that $X \in$ NP.
2. Choose a problem Y that is known to be NP-complete.
3. Prove that $Y \leq_P X$.
We noticed earlier that most of our reductions $Y \leq_P X$ consist of transforming a given instance of Y into a *single* instance of X with the same answer. This is a particular way of using a black box to solve X; in particular, it requires only a single invocation of the black box. When we use this style of reduction, we can refine the strategy above to the following outline of an NP-completeness proof.
1. Prove that $X \in$ NP.
2. Choose a problem Y that is known to be NP-complete.
3. Consider an arbitrary instance sY of problem Y, and show how to construct, in polynomial time, an instance sX of problem X that satisfies the following properties:
(a) If s_Y is a "yes" instance of Y, then s_X is a "yes" instance of X.
(b) If s_X is a "yes" instance of X, then s_Y is a "yes" instance of Y.
In other words, this establishes that s_Y and s_X have the same answer.

7. Sequencing Problems

1. The Traveling Salesman Problem

Consider a salesman who must visit n cities labeled $v_1, v_2, \ldots, v_n$. The salesman starts in city v_1, his home, and wants to find a *tour*—an order in which to visit all the other cities and return home.
His goal is to find a tour that causes him to travel as little total distance as possible.

for each ordered pair of cities (v_i, v_j), we will specify a nonnegative number $d(v_i, v_j)$ as the distance from v_i to v_j. We will not require the distance to be symmetric nor will we require it to satisfy the triangle inequality is actually less than the "direct" distance $d(v_i, v_k)$)

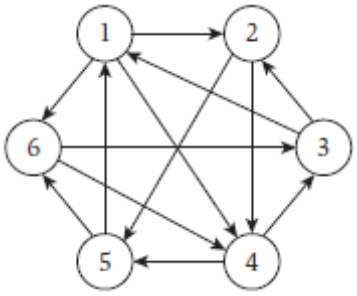
Given the set of distances, we ask: Order the cities into a *tour* $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$, with $i_1 = 1$, so as to minimize the total distance

$\sum_j d(v_{i_j}, v_{i_{j+1}}) + d(v_{i_n}, v_{i_1})$. The requirement $i_1 = 1$ simply "orients" the tour so that it starts at the home city, and the terms in the sum simply give the distance from each city on the tour to the next one. (The last term in the sum is the distance required for the salesman to return home at the end.)

2. The Hamiltonian Cycle Problem

Given a directed graph $G = (V, E)$, we say that a cycle C in G is a *Hamiltonian cycle* if it visits each vertex exactly once.

In other words, it constitutes a "tour" of all the vertices, with no repetitions.



The directed graph pictured in Figure 8.6 has several Hamiltonian cycles; one visits the nodes in the order 1, 6, 4, 3, 2, 5, 1,
while another visits the nodes in the order 1, 2, 4, 5, 6, 3, 1.

1. Proving Hamiltonian Cycle is NP-Complete

Prove that "Hamiltonian Cycle *is NP-complete".*

Proof. We first show that Hamiltonian Cycle is in NP. Given a directed graph $G = (V, E)$, a certificate that there is a solution would be the ordered list of the vertices on a Hamiltonian cycle.

Then check, in polynomial time, that this list of vertices does contain each vertex exactly once, and that each consecutive pair in the ordering is joined by an edge; this would establish that the ordering defines a Hamiltonian cycle.

We now show that 3-SAT $\leq P$ Hamiltonian Cycle. Why are we reducing from 3-SAT? Essentially, faced with Hamiltonian Cycle, we really have no idea *what* to reduce from; it's sufficiently different from all the problems we've seen so far that there's no real basis for choosing. In such a situation, one strategy is to go back to 3-SAT, since its combinatorial structure is very basic.

Of course, this strategy guarantees at least a certain level of complexity in the reduction, since we need to encode variables and clauses in the language of graphs.

So consider an arbitrary instance of 3-SAT, with variables $x1, \ldots, xn$ and clauses $C1, \ldots, Ck$. We must show how to solve it, given the ability to detect Hamiltonian cycles in directed graphs. As always, it helps to focus on the essential ingredients of 3-SAT: We can set the values of the variables however we want, and we are given three chances to satisfy each clause. We begin by describing a graph that contains $2n$ different Hamiltonian cycles that correspond very naturally to the $2n$ possible truth assignments to the variables. After this, we will add nodes to model the constraints imposed by the clauses.

We construct n paths $P1, \ldots, Pn$, where Pi consists of nodes $vi1, vi2, \ldots, vib$ for a quantity b that we take to be somewhat larger than the number of clauses k; say, $b = 3k + 3$. There are edges from vij to $vi,j+1$ and in the other direction from $vi,j+1$ to vij. Thus Pi can be traversed "left to right," from $vi1$ to vib, or "right to left," from vib to $vi1$.

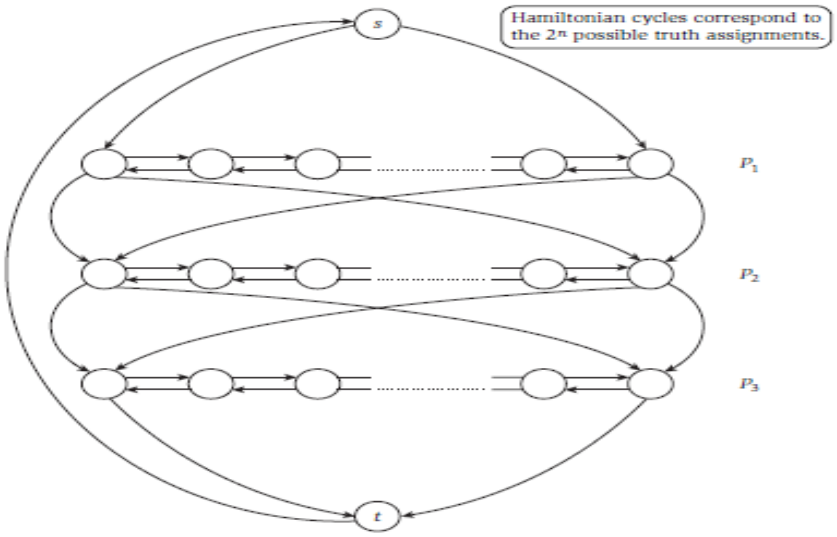


Figure 8.7 The reduction from 3-SAT to Hamiltonian Cycle: part 1.

We hook these paths together as follows. For each $i = 1, 2, \ldots, n-1$, we define edges from $vi1$ to $vi+1,1$ and to $vi+1,b$. We also define edges from vib to $vi+1,1$ and to $vi+1,b$. We add two extra nodes s and t; we define edges from s to $v11$ and $v1b$; from $vn1$ and vnb to t; and from t to s. Since only one edge leaves t, we know that any Hamiltonian cycle C must use the edge (t, s). After entering s, the cycle C can then traverse $P1$ either left to right or right to left; regardless of what it does here, it can then traverse $P2$ either left to right or right to left; and so forth, until it finishes traversing Pn and enters t. In other words, there are exactly $2n$ different Hamiltonian cycles, and they correspond to the n independent choices of how to traverse each Pi.

This naturally models the n independent choices of how to set each variables $x1, \ldots, xn$ in the 3-SAT instance. Thus we will identify each Hamiltonian cycle uniquely with a truth assignment as follows: If C traverses Pi left to right, then xi is set to 1; otherwise, xi is set to 0.

Now we add nodes to model the clauses; the 3-SAT instance will turn out to be satisfiable if and only if any Hamiltonian cycle survives. Let's consider, as a concrete example, a clause $C1 = x1 \vee x2 \vee x3$.

In the language of Hamiltonian cycles, this clause says, "The cycle should traverse $P1$ left to right; or it should traverse $P2$ right to left; or it should traverse $P3$ left to right." So we add a node $c1$, as in above figure, that does just this. (Note that certain edges have been eliminated from this drawing, for the sake of clarity.) For some value of _, node $c1$ will have edges *from* $v1\_$, $v2,\_+1$, and $v3\_$; it will have edges *to* $v1,\_+1$, $v2,\_$, and $v3,\_+1$. Thus it can be easily spliced into any Hamiltonian cycle that traverses $P1$ left to right by visiting node $c1$ between $v1\_$ and $v1,\_+1$; similarly, $c1$ can be spliced into any Hamiltonian cycle that traverses $P2$ right to left, or $P3$ left to right. It cannot be spliced into a Hamiltonian cycle that does not do any of these things.

More generally, we will define a node *cj* for each clause *Cj*.We will reserve node positions 3*j* and 3*j* + 1 in each path *Pi* for variables that participate in clause *Cj*. Suppose clause *Cj* contains a term *t*. Then if *t* = *xi*, we will add edges *(vi*,3*j* , *cj)* and *(cj* , *vi*,3*j*+1*)*; if *t* = *xi*, we will add edges *(vi*,3*j*+1, *cj)* and *(cj* , *vi*,3*j)*.

This completes the construction of the graph *G*. Now, following our generic outline for NP-completeness proofs, we claim that the 3-SAT instance is satisfiable if and only if *G* has a Hamiltonian cycle.

First suppose there is a satisfying assignment for the 3-SAT instance. Then we define a Hamiltonian cycle following our informal plan above. If *xi* is assigned 1 in the satisfying assignment, then we traverse the path *Pi* left to right; otherwise we traverse *Pi* right to left. For each clause *Cj*, since it is satisfied by the assignment, there will be at least one path *Pi* in which we will be going in the "correct" direction relative to the node *cj*, and we can splice it into the tour there via edges incident on *vi*,3*j* and *vi*,3*j*+1.

Conversely, suppose that there is a Hamiltonian cycle C in *G*. The crucial thing to observe is the following. If C enters a node *cj* on an edge from *vi*,3*j*, it must depart on an edge to *vi*,3*j*+1. For if not, then *vi*,3*j*+1 will have only one unvisited neighbor left, namely, *vi*,3*j*+2, and so the tour will not be able to visit this node and still maintain the Hamiltonian property. Symmetrically, if it enters from *vi*,3*j*+1, it must depart immediately to *vi*,3*j*. Thus, for each node *cj*, the nodes immediately before and after *cj* in the cycle C are joined by an edge *e* in *G*; thus, if we remove *cj* from the cycle and insert this edge *e* for each *j*, then we obtain a Hamiltonian cycle C on the subgraph $G - \{c1, \ldots, ck\}$. This is our original subgraph, before we added the clause nodes; as we noted above, any Hamiltonian cycle in this subgraph must traverse each *Pi* fully in one direction or the other. We thus use C to define the following truth assignment for the 3-SAT instance. If C traverses *Pi* left to right, then we set *xi* = 1; otherwise we set *xi* = 0.

Since the larger cycle C was able to visit each clause node *cj*, at least one of the paths was traversed in the "correct" direction relative to the node *cj*, and so the assignment we have defined satisfies all the clauses. Having established that the 3-SAT instance is satisfiable if and only if *G* has a Hamiltonian cycle, our proof is complete.

2. Proving Traveling Salesman is NP-Complete

Prove that Traveling Salesman is NP-complete.

Proof:

It is easy to see that Traveling Salesman is in NP: The certificate is a permutation of the cities, and a certifier checks that the length of the corresponding tour is at most the given bound. We now show that Hamiltonian Cycle $\leq_P$ Traveling Salesman. Given a directed graph *G = (V, E)*, we define the following instance of

Traveling Salesman. We have a city *v_i* for each node *vi* of the graph *G*.We define *d(v_i , v_j)* to be 1 if there is an edge *(vi , vj)* in *G*, and we define it to be 2 otherwise.

Now we claim that *G* has a Hamiltonian cycle if and only if there is tour of length at most *n* in our Traveling Salesman instance. For if *G* has a Hamiltonian cycle, then this ordering of the corresponding cities defines a tour of length *n*. Conversely, suppose there is a tour of length at most *n*. The expression for the length of this tour is a sum of *n* terms, each of which is at least 1; thus it must be the case that all the terms are equal to 1. Hence each pair of nodes in *G* that correspond to consecutive cities on the tour must be connected by an edge; it follows that the ordering of these corresponding nodes must form a Hamiltonian cycle.

Prove that Hamiltonian Path is NP-complete.

Proof:

First of all, Hamiltonian Path is in NP: A certificate could be a path in *G*, and a certifier could then check that it is indeed a path and that it contains each node exactly once. One way to show that Hamiltonian Path is NP-complete is to use a reduction from 3-SAT that is almost identical to the one we used for Hamiltonian Cycle: We construct the same graph that appears in above figure, *except* that we

do not include an edge from *t* to *s*. If there is any Hamiltonian path in this modified graph, it must begin at *s* (since *s* has no incoming edges) and end at *t* (since *t* has no outgoing edges). With this one change, we can adapt the argument used in the Hamiltonian Cycle reduction more or less word for word to argue that there is a satisfying assignment for the instance of 3-SAT if and only if there is a Hamiltonian path.

An alternate way to show that Hamiltonian Path is NP-complete is to prove that Hamiltonian Cycle $\leq P$ Hamiltonian Path. Given an instance of Hamiltonian.

Cycle, specified by a directed graph *G*, we construct a graph *G*_ as follows. We choose an arbitrary node *v* in *G* and replace it with two new nodes *v*_ and *v*''. All edges out of *v* in *G* are now out of *v*'; and all edges into *v* in *G* are now into *v*''. More precisely, each edge *(v, w)* in *G* is replaced by an edge *(v', w)*; and each edge *(u, v)* in *G* is replaced by an edge *(u, v'')*. This completes the construction of *G*'.

We claim that *G*' contains a Hamiltonian path if and only if *G* contains a Hamiltonian cycle. Indeed, suppose *C* is a Hamiltonian cycle in *G*, and consider traversing it beginning and ending at node *v*. It is easy to see that the same ordering of nodes forms a Hamiltonian path in *G*_ that begins at *v*_ and ends at *V*''. Conversely, suppose *P* is a Hamiltonian path in *G*'. Clearly *P* must begin at *v*' (since *v*' has no incoming edges) and end at *v*__ (since *v*__ has no outgoing edges). If we replace *v*_ and *v*__ with *v*, then this ordering of nodes forms a Hamiltonian cycle in *G*.