CHAPTER

13 APPROXIMABILITY

Although all NP-complete problems share the same worst-case complexity, they have little else in common. When seen from almost any other perspective, they resume their healthy, confusing diversity. Approximability is a case in point.

13.1 APPROXIMATION ALGORITHMS

An **NP**-completeness proof is typically the first act of the analysis of a computational problem by the methods of the theory of algorithms and complexity, not the last. Once **NP**-completeness has been established, we are motivated to explore possibilities that are less ambitious than solving the problem exactly, efficiently, every time. If we are dealing with an optimization problem, we may want to study the behavior of *heuristics*, "quick-and-dirty" algorithms which return feasible solutions that are not necessarily optimal. Such heuristics can be empirically valuable methods for attacking an **NP**-complete optimization problem even when nothing can be proved about their worst-case (or expected) performance. In some fortunate cases, however, the solutions returned by a polynomial-time heuristic are guaranteed to be "not too far from the optimum." We formalize this below:

Definition 13.1: Suppose that A is an optimization problem; this means that for each instance x we have a set of *feasible solutions*, call it $F(x)$, and for each such solution $s \in F(x)$ we have a positive integer cost $c(s)$ (we use the term *cost* and the notation $c(s)$ even in the case of maximization problems). The optimum cost is defined then as $\mathrm{OPT}(x) = \min_{s \in F(x)} c(s)$ (or $\max_{s \in F(x)} c(s)$, if A is a maximization problem). Let M be an algorithm which, given any

instance x, returns a feasible solution $M(x) \in F(x)$. We say that M is an *ϵ-approximation algorithm*, where $\epsilon \geq 0$, if for all x we have

$$\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} \leq \epsilon.$$

Recall that all costs are assumed to be positive, and thus this ratio is always well-defined. Thus, a heuristic is ϵ-approximate if, intuitively, the "relative error" of the solution found is at most ϵ. We use $\max\{\mathrm{OPT}(x), c(M(x))\}$ in the denominator, instead of the more natural $\mathrm{OPT}(x)$, in order to make the definition symmetric with respect to minimization and maximization problems: This way, for both kinds of problems ϵ takes values between 0 and 1. For maximization problems, an ϵ-approximate algorithm returns solutions that are never smaller than $1 - \epsilon$ times the optimum. For minimization problems, the solution returned is never more than $\frac{1}{1-\epsilon}$ times the optimum. □

For each **NP**-complete optimization problem A we shall be interested in determining the smallest ϵ for which there is a *polynomial-time* ϵ-approximation algorithm for A. Sometimes no such smallest ϵ exists, but there are approximation algorithms that achieve arbitrarily small error ratios (we see an example in the next section).

Definition 13.2: The *approximation threshold* of A is the greatest lower bound of all $\epsilon > 0$ such that there is a polynomial-time ϵ-approximation algorithm for A. □

The approximation threshold of an optimization problem, minimization *or* maximization, can be anywhere between zero (arbitrarily close approximation is possible) and one (essentially no approximation is possible). Of course, for all we know $\mathbf{P} = \mathbf{NP}$, and thus all optimization problems in **NP**, have approximation threshold zero. It turns out that **NP**-complete optimization problems behave in very diverse and intriguing ways with respect to this important parameter—because reductions typically fail to preserve the approximation threshold of problems. We turn immediately to some examples.

Node Cover

NODE COVER (Corollary 2 to Theorem 9.4) is an **NP**-complete minimization problem, where we seek the smallest set of nodes $C \subseteq V$ in a graph $G = (V, E)$, such that for each edge in E at least one of its endpoints is in C.

What is a plausible heuristic for obtaining a "good" node cover? Here is a first try: If a node v has high degree, it is obviously useful for covering many edges, and so it is probably a good idea to add it to the cover. This suggests the following "greedy" heuristic:

Start with $C = \emptyset$. While there are still edges left in G, choose the node in G with the largest degree, add it to C, and delete it from G.

As it turns out, this heuristic is *not* an ϵ-approximation algorithm, for *any* $\epsilon < 1$—its error ratio grows as $\log n$ (see Problem 13.4.1), where n is the of the number of nodes of G, and thus no ϵ smaller than 1 is valid.

In order to achieve a decent approximation of NODE COVER, we must employ a technique that appears even less sophisticated than the greedy heuristic:

Start with $C = \emptyset$. While there are still edges left in G, choose any edge $[u, v]$, add both u and v to C, and delete them from G.

Suppose that this heuristic ends up with a node cover C. How far off the optimum can C be? Notice that C contains $\frac{1}{2}|C|$ edges of G, no two of which share a node (a *matching*). Any node cover, *including the optimum one*, must contain at least one node from each of these edges (otherwise, an edge would not be covered). It follows that $\text{OPT}(G) \geq \frac{1}{2}|C|$, and thus $\frac{|C|-\text{OPT}(G)}{|C|} \leq \frac{1}{2}$. We have shown the following:

Theorem 13.1: The approximation threshold of NODE COVER is at most $\frac{1}{2}$. □

Surprisingly, this simple algorithm is the best approximation algorithm known for NODE COVER.

Maximum Satisfiability

In MAXSAT we are given a set of clauses, and we seek the truth assignment that satisfies the most. The problem is **NP**-complete even if the clauses have at most two literals (recall Theorem 9.2).

Our approximation algorithm for MAXSAT is best described in terms of a more general problem called k-MAXGSAT (for maximum *generalized* satisfiability). In this problem we are given a set of Boolean expressions $\Phi = \{\phi_1, \ldots, \phi_m\}$ in n variables, where each expression is not necessarily a disjunction of literals as in MAXSAT, but is a general Boolean expression involving at most k of the n Boolean variables, where $k > 0$ is a fixed constant (we can in fact assume for simplicity that each expression involves *exactly* k variables, some of which may not be explicitly mentioned in it). We are seeking the truth assignment that satisfies the most expressions.

Although our approximation algorithm for this problem will be perfectly deterministic, it is best motivated by a probabilistic consideration. Suppose that we pick one of the 2^n truth assignments at random. How many expressions in Φ should we expect to satisfy? The answer is easy to calculate. Each expression $\phi_i \in \Phi$ involves k Boolean variables. Out of the 2^k truth assignments, we can easily calculate the number t_i of truth assignments that satisfy ϕ_i. Thus, a random truth assignment will satisfy ϕ_i with probability $p(\phi_i) = \frac{t_i}{2^k}$. The expected number of satisfied expressions is simply the sum of these probabilities: $p(\Phi) = \sum_{i=1}^{m} p(\phi_i)$.

Suppose that we set $x_1 =$ **true** in all expressions of Φ; a set of expressions

$\Phi[x_1 = \textbf{true}]$ involving the variables $x_2, \ldots, x_n$ results, and we can again calculate $p(\Phi[x_i = \textbf{true}])$. Similarly for $p(\Phi[x_1 = \textbf{false}])$. Now it is very easy to see that

$$p(\Phi) = \frac{1}{2}(p(\Phi[x_1 = \textbf{true}]) + p(\Phi[x_1 = \textbf{false}])).$$

This equation means that, if we modify Φ by setting x_1 equal to the truth value t that yields the largest $p(\Phi[x_1 = t])$, *we end up with an expression set with expectation at least as large as the original.*

We can continue like this, always assigning to the next variable the value that maximizes the expectation of the resulting expression set. In the end, all variables have been given values, and all expressions are either **true** (have been satisfied) or **false** (have been falsified). However, since our expectation never decreased in the process, we know that at least $p(\Phi)$ expressions have been satisfied.

Thus, our algorithm satisfies at least $p(\Phi)$ expressions. Since the optimum cannot be more than the total number of expressions in Φ that are *individually satisfiable* (that is, those for which $p(\phi_i) > 0$), the ratio is at least equal to the smallest positive $p(\phi_i)$—recall that $p(\Phi)$ is the sum of all these positive $p(\phi_i)$'s. We conclude that the above heuristic is a polynomial-time ϵ-approximation algorithm for k-MAXGSAT, *where ϵ is one minus the smallest probability of satisfaction of any satisfiable formula in Φ.* For any satisfiable expression ϕ_i involving k Boolean variables, this probability is at least 2^{-k} (since at least one of the 2^k possible truth assignments on the k variables must satisfy the expression) and thus this algorithm is ϵ-approximate with $\epsilon = 1 - 2^{-k}$.

Now if the ϕ_i's are clauses (this brings us back to MAXSAT), then the situation is far better: The probability of satisfaction is at least $\frac{1}{2}$, and $\epsilon = \frac{1}{2}$. If we restrict the clauses to have *at least* k distinct literals (notice the reversal in the usual restriction), then the probability that a random truth assignment satisfies a clause is obviously $1-2^{-k}$ (all truth assignments are satisfying, except for the one that makes all literals **false**), and the approximation ratio becomes $\epsilon = 2^{-k}$.

We can summarize our discussion of maximum satisfiability problems thus:

Theorem 13.2: The approximation threshold of k-MAXGSAT is at most $1-2^{-k}$. The approximation threshold of MAXSAT (the special case of MAXGSAT where all expressions are clauses) is at most $\frac{1}{2}$; and when each clause has at least k distinct literals, the approximation threshold of the resulting problem is at most 2^{-k}. □

These are the best polynomial-time approximation algorithms known for k-MAXGSAT and MAXSAT with at least k literals per clause; the best upper bound known for the approximation threshold of general MAXSAT is $\frac{1}{4}$.

Maximum Cut

In MAX-CUT we want to partition the nodes of $G = (V, E)$ into two sets S and $V - S$ such that there are as many edges as possible between S and $V - S$; MAX-CUT is **NP**-complete (Theorem 9.5).

An interesting approximation algorithm for MAX-CUT is based on the idea of *local improvement* (recall Example 10.6). We start from any partition of the nodes of $G = (V, E)$ (even $S = \emptyset$), and repeat the following step: If the cut can be made larger (more edges would be in it) by adding a single node to S, or by deleting a single node from S, then we do so. If no improvement is possible, we stop and return the cut thus obtained.

One can develop such local improvement algorithms for just about any optimization problem. Sometimes such heuristics are extremely useful, but usually very little can be proved about their performance —both the time required (recall Example 10.6) and the ratio to the optimum. Fortunately, the present case is an exception. First notice that, since the maximum cut can have at most $|E|$ edges, and each local improvement adds at least one edge to the cut, the algorithm must end after at most $|E|$ improvements. (In general, any local improvement algorithm in an optimization problem with polynomially bounded costs will be polynomial.) Furthermore, *we claim that the cut resulting from this algorithm has at least half as many edges as the optimum*, and thus this simple local improvement heuristic is a polynomial-time $\frac{1}{2}$-approximation algorithm for MAX-CUT.

In proof, consider a decomposition of V into four disjoint subsets $V = V_1 \cup V_2 \cup V_3 \cup V_4$, such that the partition obtained by our heuristic is $(V_1 \cup V_2, V_3 \cup V_4)$, whereas the optimum partition is $(V_1 \cup V_3, V_2 \cup V_4)$. Let e_{ij}, with $1 \leq i \leq j \leq 4$ be the number of edges between node sets V_i and V_j (see Figure 13.1). All we know about our partition is that it cannot be improved by migrating any node in the other set. Thus, for each node in V_1, its edges to V_1 and V_2 are outnumbered by those to V_3 and V_4. Considering now all nodes in V_1 together we obtain $2e_{11} + e_{12} \leq e_{13} + e_{14}$, from which we conclude $e_{12} \leq e_{13} + e_{14}$ Similarly we can get the following inequalities, by considering the other three sets of nodes:

$$e_{12} \leq e_{23} + e_{24}$$

$$e_{34} \leq e_{23} + e_{13}$$

$$e_{34} \leq e_{14} + e_{24}.$$

Adding all these inequalities, dividing both sides by two, and adding the inequality $e_{14} + e_{23} \leq e_{14} + e_{23} + e_{13} + e_{24}$ we obtain

$$e_{12} + e_{34} + e_{14} + e_{23} \leq 2 \cdot (e_{13} + e_{14} + e_{23} + e_{24}),$$

which is the same as saying that our solution is at least half the optimum. We have shown:

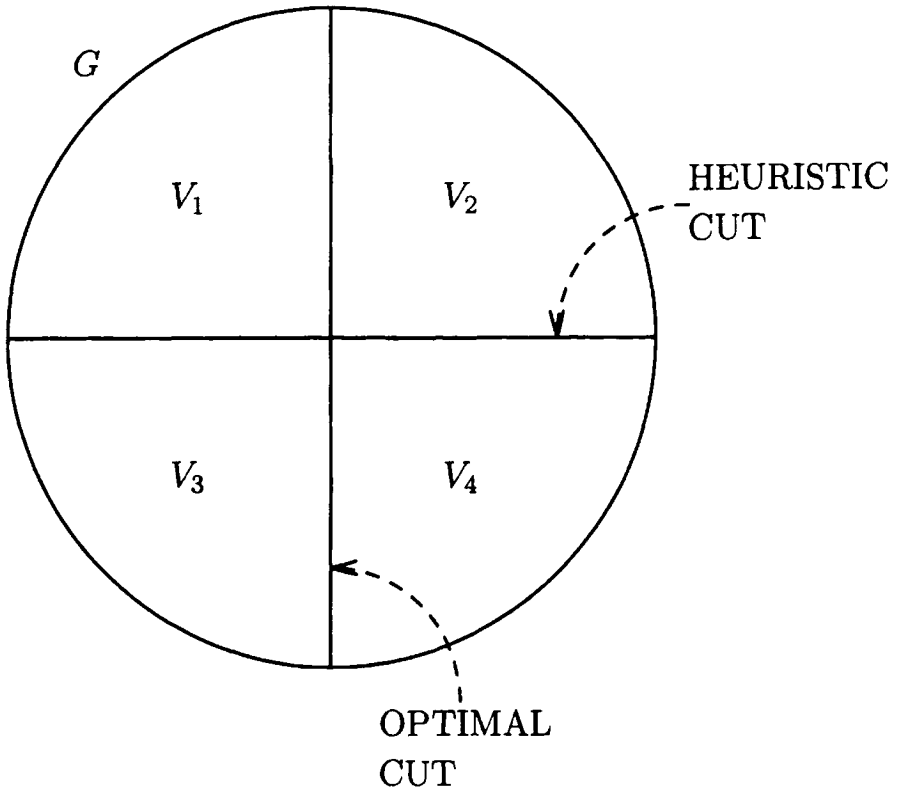

Figure 13-1. The argument for MAX-CUT.

Theorem 13.3: The approximation threshold of MAX-CUT is at most $\frac{1}{2}$. □

The Traveling Salesman Problem

In all three cases of NODE COVER, MAXSAT, and MAX-CUT that we have seen so far, we have exhibited algorithms that guarantee some approximation threshold strictly less than one. For the TSP the situation is very bleak in comparison: If there is a polynomial-time ϵ-approximation algorithm for the TSP for any $\epsilon < 1$, then it follows that **P** = **NP**—and approximation algorithms are pointless...

Theorem 13.4: Unless **P** = **NP**, the approximation threshold of the TSP is one.

Proof: Suppose that there is a polynomial-time ϵ-approximation algorithm for the TSP for some $\epsilon < 1$. Using this algorithm, we shall derive a polynomial-time algorithm for the **NP**-complete problem HAMILTON CYCLE (recall the Theorem 9.7 and Problem 9.5.15). Notice that this would conclude the proof.

Given any graph $G = (V, E)$, our algorithm for HAMILTON CYCLE constructs an instance of the TSP with $|V|$ cities. The distance between city i and j is one if there is an edge between nodes i and j in G, and it is $\frac{|V|}{1-\epsilon}$ if there is no $[i, j]$ edge in E. Having constructed this instance of the TSP, we next apply our hypothetical polynomial-time ϵ-approximation algorithm to it. There are two cases: If the algorithm returns a tour of total cost $|V|$—that is, with only

unit-length edges—then we know that G has a Hamilton cycle. If on the other hand the algorithm returns a tour with at least one edge of length $\frac{|V|}{1-\epsilon}$, then the total length of this tour is strictly greater than $\frac{|V|}{1-\epsilon}$. Since we have assumed that our algorithm is ϵ-approximate, that is, the optimum is never less than $1-\epsilon$ times the solution returned, we must conclude that the optimum tour has cost greater than $|V|$, and thus G has no Hamilton cycle. Thus, we can decide whether a graph has a Hamilton cycle simply by creating the instance of the TSP as described, and running the hypothetical ϵ-approximate algorithm on it. □

Notice the specialized kind of reduction employed in this proof of impossibility: In the constructed instance there is a large "gap" between the optimum cost when the original instance is a "yes" instance of the Hamilton cycle problem, and the optimum cost when the original instance is a "no" instance. It is then shown that an approximation algorithm could detect such a gap.

As usual, a negative result for a problem may not hold for its special cases. Let us consider the special case of the TSP in which all distances are either 1 or 2 (this is the special case that we proved **NP**-complete in Corollary to Theorem 9.7). It is amusing to notice that in this case, *any* algorithm is $\frac{1}{2}$-approximate—because all tours have length at most twice the optimum! But we can do much better: There is a polynomial-time $\frac{1}{7}$-approximation algorithm for this problem (see the references in 13.4.8). Even in the more general case in which the distances are not quite all ones and twos, but they *satisfy the triangle inequality* $d_{ij} + d_{jk} \leq d_{ik}$, there is a very simple and clever polynomial-time $\frac{1}{3}$-approximation algorithm (see the references in 13.4.8). In both cases, we know of no better approximation algorithms.

Knapsack

We have seen several optimization problems (MAXSAT, NODE COVER, MAX-CUT, TSP with distances 1 and 2) for which ϵ-approximation exists for some ϵ (and it is open whether smaller ϵ's are achievable), and the general TSP for which no ϵ-approximation is possible unless **P** = **NP**. KNAPSACK is an optimization problem for which approximability has no limits:

Theorem 13.5: The approximation threshold of KNAPSACK is 0. That is, for any $\epsilon > 0$ there is a polynomial ϵ-approximation algorithm for KNAPSACK.

Proof: Suppose that we are given an instance x of KNAPSACK. That is, we have n weights $w_i, i = 1, \ldots, n$, a weight limit W, and n values $v_i, i = 1, \ldots, n$. We must find a subset $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is the largest possible.

We have seen in Section 9.4 that there is a *pseudopolynomial* algorithm for KNAPSACK, one that works in time proportional to *the weights* in the instance. We develop now a *dual* approach, one that works with the values instead of the

weights. Let $V = \max\{v_1, \ldots, v_n\}$ be the maximum value, and let us define for each $i = 0, 1, \ldots, n$ and $0 \leq v \leq nV$ the quantity $W(i, v)$ to be *the minimum weight attainable by selecting some among the i first items, so that their value is exactly v*. We start with $W(0, v) = \infty$ for all i and v, and then

$$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}.$$

In the end, we pick the largest v such that $W(n, v) \leq W$. Obviously, this algorithm solves KNAPSACK in time $\mathcal{O}(n^2 V)$.

But of course the values may be huge integers, and this algorithm is a pseudopolynomial, not a polynomial one. However, now that we are only interested in *approximating* the optimum value, a maneuver suggests itself: We may want to disregard the last few bits of the values, thus *trading off accuracy for speed*. Given the instance $x = (w_1, \ldots, w_n, W, v_1, \ldots, v_n)$, we can define the *approximate* instance $x' = (w_1, \ldots, w_n, W, v'_1, \ldots, v'_n)$, where the new values are $v'_i = 2^b \lfloor \frac{v_i}{2^b} \rfloor$, the old values with their b least significant bits replaced by zeros; b is an important parameter to be fixed in good time.

If we solve the approximate instance x' instead of x, the time required will be only $\mathcal{O}(\frac{n^2 V}{2^b})$, because we can ignore the trailing zeros in the v'_i's. The solution S' obtained will in general be different from the optimum solution S of x, but the following sequence of inequalities shows that their values cannot be very far:

$$\sum_{i \in S} v_i \geq \sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S} (v_i - 2^b) \geq \sum_{i \in S} v_i - n2^b$$

The first inequality holds because S is optimum in x, the second because $v'_i \leq v_i$, the next because S' is optimum in x', the next because $v'_i \geq v_i - 2^b$, and the last because $|S| \leq n$. Comparing the second and last expression, we conclude that the value of the solution returned by our approximation algorithm is at most $n2^b$ below the optimum. Since V is a lower bound on the value of the optimum solution (assume with no loss of generality that that $w_i \leq W$ for all i), the relative deviation from the optimum is at most $\epsilon = \frac{n2^b}{V}$.

We are at a remarkably favorable position: Given *any* user-supplied $\epsilon > 0$, we can truncate the last $b = \lceil \log \frac{\epsilon V}{n} \rceil$ bits of the values, and arrive at an ϵ-approximation algorithm with running time $\mathcal{O}(\frac{n^2 V}{2^b}) = \mathcal{O}(\frac{n^3}{\epsilon})$—a polynomial! We conclude that there is a polynomial-time ϵ-approximation algorithm for any $\epsilon > 0$. Consequently, the greatest lower bound of all achievable ratios is zero. □

Any problem with zero approximation threshold, such as KNAPSACK, has a sequence of algorithms whose error ratios have limit 0. In the case of KNAPSACK the sequence is especially well-behaved, in that the algorithms in the sequence can be seen as the same algorithm supplied with different ϵ's.

Definition 13.3: A *polynomial-time approximation scheme* for an optimization problem A is an algorithm which, for each $\epsilon > 0$ and instance x of A, returns a solution with a relative error of at most ϵ, in time which is bounded by a polynomial (depending on ϵ) of $|x|$. In the case of KNAPSACK, where the polynomial depends *polynomially* on $\frac{1}{\epsilon}$ as well—recall the $\mathcal{O}(\frac{n^3}{\epsilon})$ bound— the approximation scheme is called *fully polynomial.* □

Not all polynomial-time approximation schemes need be fully polynomial. For example, no strongly **NP**-complete optimization problem can have a fully polynomial-time approximation scheme, unless **P** = **NP** (see Problem 13.4.2). For example, there is a polynomial-time approximation scheme for BIN PACKING (which is strongly **NP**-complete, recall Theorem 9.11) whose time bound depends exponentially on $\frac{1}{\epsilon}$ (see Problem 13.4.6). For another example of such a scheme, see the next subsection.

Maximum Independent Set

We have seen optimization problems all over the approximability spectrum. Some problems such as the TSP have approximation threshold one (unless **P** = **NP**); others like KNAPSACK have approximation threshold zero; and still others (NODE COVER, MAXSAT, etc.) seem to be in between, with an approximation threshold which is known to be strictly less than one, but not known to be zero. We shall next prove that the INDEPENDENT SET problem belongs in one of the two extreme classes: *Its approximation threshold is either zero or one* (later in this chapter we shall see which of the two extremes is the true answer).

This result is shown by a *product construction.* Let $G = (V, E)$ be a graph. The *square* of G, G^2, is a graph with vertices $V \times V$, and the edges $\{[(u,u'),(v,v')] :$ either $u = v$ and $[u',v'] \in E$, or $[u,v] \in E\}$. See Figure 13.2 for an example. The crucial property of G^2 is this:

Lemma 13.1: G has an independent set of size k if and only if G^2 has an independent set of size k^2.

Proof: If G has an independent set $I \subseteq V$ with $|I| = k$, then the following is an independent set of G^2 of size k^2: $\{(u,v) : u,v \in I\}$. Conversely, if I^2 is an independent set of G^2 with k^2 vertices, then both $\{u : (u,v) \in I^2$ for some $v \in V\}$ or $\{v : (u,v) \in I^2$ for some $u \in V\}$ are independent sets of G, and one of them must have at least k vertices. □

From this we can show:

Theorem 13.6: If there is an ϵ_0-approximation algorithm for INDEPENDENT SET for *any* $\epsilon_0 < 1$, then there is a polynomial-time approximation scheme for INDEPENDENT SET.

Proof: Suppose that a $\mathcal{O}(n^k)$ time-bounded ϵ_0-approximation algorithm exists, and we are given a graph G. If we apply this algorithm to G^2, we obtain in time

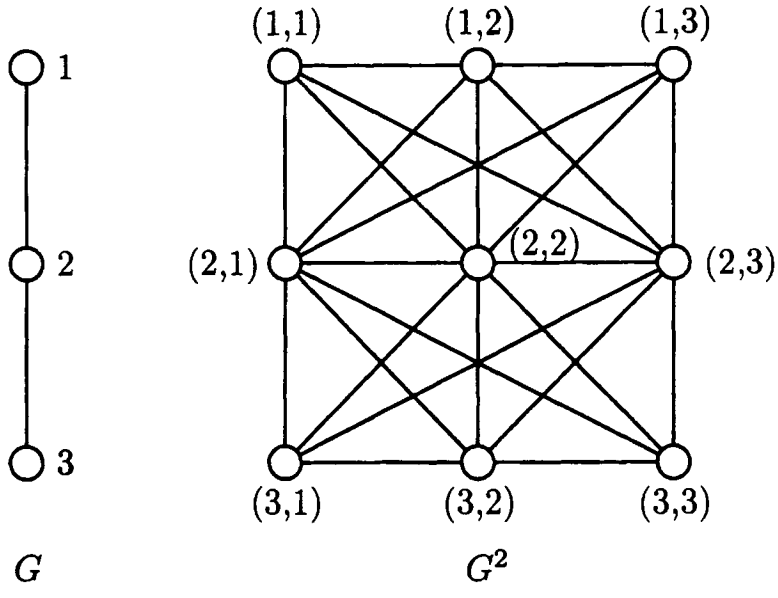

Figure 13-2. The product construction.

$\mathcal{O}(n^{2k})$ an independent set of size at least $(1-\epsilon_0)\cdot k^2$, where k is the maximum independent set of G (and thus k^2 is the maximum independent set of G^2). From this, by the construction in the Lemma, we can get an independent set of G equal to at least the square root of $(1-\epsilon_0)\cdot k^2$, or $\sqrt{1-\epsilon_0}\cdot k$. That is, if we have an ϵ_0-approximation algorithm for INDEPENDENT SET, then we have an ϵ_1-approximation algorithm, where $\epsilon_1 = 1-\sqrt{1-\epsilon_0}$.

Thus, if we have an ϵ_0-approximation algorithm, then the product construction yields an ϵ_1-approximation algorithm. If we apply the product construction twice (that is, apply our approximation algorithm to $(G^2)^2$) then we have an ϵ_2-approximation algorithm, where $1-\epsilon_2 = \sqrt[4]{1-\epsilon_0}$. And so on.

For any given $\epsilon > 0$, however small, we can repeat the product construction $\ell = \lceil \log \frac{\log(1-\epsilon_0)}{\log(1-\epsilon)} \rceil$ times. We obtain an algorithm with time bound $\mathcal{O}(n^{2^\ell k}) = \mathcal{O}(n^{k\frac{\log(1-\epsilon_0)}{\log(1-\epsilon)}})$, and approximation ratio at most ϵ, the desired polynomial-time approximation scheme. (Notice that this time bound is *not* a polynomial in n and $\frac{1}{\epsilon}$, and hence this (hypothetical) polynomial-time approximation scheme would not be fully polynomial.) □

It is instructive at this point to contrast the approximability of the INDEPENDENT SET and NODE COVER problems, two problems that are reducible to each other in a remarkably trivial way (recall Corollary 1 to Theorem 9.3).

Another interesting post scriptum to the approximability of INDEPENDENT SET: If we restrict our graphs so that no node has degree greater than four, then the resulting special case, which we may call k-DEGREE INDEPENDENT SET, remains **NP**-complete (Corollary 1 to Theorem 9.4). However, a simple approximation algorithm is now possible.

We start with $I = \emptyset$. While there are nodes left in G we repeatedly delete from G any node v and all of its adjacent nodes, adding v to I. Obviously, the resulting I will be an independent set of G. Since each stage of the algorithm adds another node to I and deletes at most $k+1$ nodes from G (the node added and its neighbors, at most k of them), the resulting independent set has at least $\frac{|V|}{k+1}$ nodes, which is at least $\frac{1}{k+1}$ times the true maximum independent set. We have shown:

Theorem 13.7: The approximation threshold of the k-DEGREE INDEPENDENT SET problem is at most $\frac{k}{k+1}$. □

Once again, no better polynomial-time approximation algorithm for this problem is known.

13.2 APPROXIMATION AND COMPLEXITY

A polynomial-time approximation scheme for an optimization problem is rightfully considered the next best thing to a polynomial-time *exact* algorithm for the problem. For **NP**-complete optimization problems an important question is whether such a scheme exists. Since individual questions of this sort are so hard to answer, in this section we do something that parallels the development of **NP**-completeness: We lump many such problems together by reductions so that they are all complete for a natural and meaningful complexity class.

L-Reductions

We have seen in several occasions that ordinary reductions are grossly inadequate for studying approximability. We next introduce a careful kind of reduction that preserves approximability.

Definition 13.4: Optimization problems are certainly function problems (since the optimum solution, and not just a "yes" or "no" answer, is sought). Recall from Definition 10.1 that a reduction between function problems A and B is a pair of functions R and S, where R is computable in logarithmic space and S in polynomial time, such that if x is an instance of A then $R(x)$ is an instance of B; and furthermore, if y is an answer of $R(x)$ then $S(y)$ is an answer of x.

Suppose that A and B are optimization problems (maximization *or* minimization). An *L-reduction* from A to B is a pair of functions R and S, both computable in logarithmic space, with the following two additional properties: First, if x is an instance of A with optimum cost $\mathrm{OPT}(x)$, then $R(x)$ is an instance of B with optimum cost that satisfies

$$\mathrm{OPT}(R(x)) \leq \alpha \cdot \mathrm{OPT}(x),$$

where α is a positive constant. Second, if s is any feasible solution of $R(x)$,

then $S(s)$ is a feasible solution of x such that

$$|\mathrm{OPT}(x) - c(S(s))| \leq \beta \cdot |\mathrm{OPT}(R(x)) - c(s)|,$$

where β is another positive constant particular to the reduction (and we use c to denote the cost in both instances). That is, S is guaranteed to return a feasible solution of x which is not much more suboptimal than the given solution of $R(x)$. Notice that, by the second property, an L-reduction is a true reduction: If s is the optimum solution of $R(x)$, then indeed $S(s)$ must the optimum solution of x. □

Example 13.1: The trivial reduction from INDEPENDENT SET to NODE COVER (R is the identity function, returning the same graph G, while S replaces C with $V - C$) is not an L-reduction. Its flaw is that the optimum node cover may be arbitrarily larger than the optimum independent set (consider the case in which G is a large clique), violating the first condition.

However, if we restrict our graphs to have maximum degree k, the problems go away, and (R, S) is indeed an L-reduction from k-DEGREE INDEPENDENT SET to k-DEGREE NODE COVER. In proof, if the maximum degree is k then the maximum independent set is at least $\frac{|V|}{k+1}$, while the minimum node cover has at most $|V|$ nodes, and so the constant $\alpha = k + 1$ satisfies the first condition. And the difference between any cover C and the optimum is the same as the difference between $V - C$ and the maximum independent set, and hence we can take $\beta = 1$ in the second condition.

Similarly, it is easy to see that (R, S) is also an L-reduction in the opposite direction, with the same α and β. □

L-reductions have the important composition property of ordinary reductions (Proposition 8.2):

Proposition 13.1: If (R, S) is an L-reduction from problem A to problem B, and (R', S') is an L-reduction from B to C, then their composition $(R \cdot R', S' \cdot S)$ is an L-reduction from A to C.

Proof: It follows from Proposition 8.2 that $R \cdot R'$ and $S' \cdot S$ are computable in logarithmic space. Also, if x is an instance of A, we have $\mathrm{OPT}(x) \leq \alpha\mathrm{OPT}(R(x))$ and $\mathrm{OPT}(R(x)) \leq \alpha\mathrm{OPT}(R'(R(x)))$, where α and α' are the corresponding constants, and therefore $\mathrm{OPT}(x) \leq \alpha \cdot \alpha'\mathrm{OPT}(R'(R(x)))$ and $R \cdot R'$ satisfies the first condition with the constant $\alpha \cdot \alpha'$. Similarly, it is easy to check that $S' \cdot S$ satisfies the second condition with the constant $\beta \cdot \beta'$. □

The key property of L-reductions is that they preserve approximability:

Proposition 13.2: If there is an L-reduction (R, S) from A to B with constants α and β, and there is a polynomial-time ϵ-approximation algorithm for B, then there is a polynomial-time $\frac{\alpha\beta\epsilon}{1-\epsilon}$-approximation algorithm for A.

Proof: The algorithm is this: Given an instance x of A, we construct the instance $R(x)$ of B, and then apply to it the assumed ϵ-approximation algorithm for B to obtain solution s. Finally, we compute the solution $S(s)$ of A. We claim that this is an $\frac{\alpha\beta\epsilon}{1-\epsilon}$-approximation algorithm for A.

In proof, consider the ratio $\frac{|\mathrm{OPT}(x)-c(S(s))|}{\max\{\mathrm{OPT}(x),c(S(s))\}}$. By the second property of L-reductions, the numerator is at most $\beta|\mathrm{OPT}(R(x)) - c(s)|$. By the first property of L-reductions, the denominator is at least $\frac{\mathrm{OPT}(R(x))}{\alpha}$, which is at least $\frac{\max\{\mathrm{OPT}(R(x)),c(s)\}}{\alpha}(1-\epsilon)$. Dividing the two inequalities we conclude that $\frac{|\mathrm{OPT}(x)-x(S(s))|}{\max\{\mathrm{OPT}(x),c(S(s))\}} \leq \frac{\alpha\beta}{1-\epsilon}\frac{|\mathrm{OPT}(R(x))-c(s)|}{\max\{\mathrm{OPT}(R(x)),c(s)\}}$. The latter quantity is at most $\frac{\alpha\beta\epsilon}{1-\epsilon}$ by our hypothesis about the ϵ-approximation algorithm for B. □

The important property of the expression $\frac{\alpha\beta\epsilon}{1-\epsilon}$ in the statement of Proposition 13.2 is this: *If ϵ tends to zero from positive values, then so does the expression.* This implies the following:

Corollary: If there is an L-reduction from A to B and there is a polynomial-time approximation scheme for B, then there is a polynomial-time approximation scheme for A. □

The Class MAXSNP

Our development of a complexity theory of approximability parallels the reasoning that led us to the **P** $\neq$ **NP** conjecture. In both cases, for several natural problems we asked an important and difficult-to-answer question (then whether they have a polynomial-time algorithm, now whether they have a polynomial-time approximation scheme). We next defined a concept of reduction that preserves the property in question. To proceed we need the analog of **NP**, a broad class of problems, that contains many important complete ones. We define it next.

Definition 13.5: Our motivation for the next definition comes from Fagin's Theorem, stating that all graph-theoretic properties in **NP** can be expressed in existential second-order logic (Theorem 8.3). There is an interesting fragment of **NP**, called *strict* **NP** or **SNP**, which consists of all properties expressible as

$$\exists S \forall x_1 \forall x_2 \ldots \forall x_k \phi(S, G, x_1, \ldots, x_k),$$

where ϕ is a quantifier-free First-Order expression involving the variables x_i and the structures G (the input) and S. **NP** is more general than **SNP**, in that it allows arbitrary first-order quantifiers, not just universal ones.

Naturally, **SNP** contains decision problems, and we are presently interested in defining a class of *optimization* problems. There is a simple and interesting way to obtain a broad class of optimization problems by modifying the **SNP** expressions. Consider any such expression $\exists S \forall x_1 \forall x_2 \ldots \forall x_k \phi$. It asks for a

relation S such that all possible k-tuples of nodes $(x_1, \ldots, x_k)$ satisfy ϕ. Suppose that we compromise a little. Instead of requiring that ϕ hold *for all* k-tuples, we seek the relation S such that ϕ holds for *as many* k-*tuples* $(x_1, \ldots, x_k)$ *as possible*. We thus arrive at an optimization problem. We generalize the situation slightly, so that the input structure G is not necessarily a single binary relation, but a collection $G_1, \ldots, G_m$ of relations of arbitrary arity.

Define now **MAXSNP$_0$** (not yet our final goal) to be the following class of optimization problems: Problem A in this class is defined in terms of the expression

$$\max_S |\{(x_1, \ldots, x_k) \in V^k : \phi(G_1, \ldots, G_m, S, x_1, \ldots, x_k)\}|$$

(compare with $\exists S \forall x_1 \forall x_2 \ldots \forall x_k \phi$). The input to a problem A is a set of relations $G_1, \ldots, G_m$ over a finite universe V. We are seeking a relation $S \subseteq V^r$ such that the number of k-tuples $(x_1, \ldots, x_k)$ for which ϕ holds is as large as possible. Notice how the expression that defines A is derived from the original $\exists S \forall x_1 \forall x_2 \ldots \forall x_k \phi$. The existential quantifier now seeks the maximizing S, while the sequence of k universal quantifiers has become a counter of k-tuples.

Finally, define **MAXSNP** to be the class of all optimization problems that are L-reducible to a problem in **MAXSNP$_0$**. □

Example 13.2: The problem MAX-CUT is in **MAXSNP$_0$**, and therefore in **MAXSNP**. In proof, MAX-CUT can be written as follows:

$$\max_{S \subseteq V} |\{(x, y) : ((G(x, y) \vee G(y, x)) \wedge S(x) \wedge \neg S(y))\}|.$$

Here we represent a graph as a directed graph, assigning to each undirected edge an arbitrary direction. The problem as stated asks for the subset S of nodes that maximizes the number of edges that enter S or leave S; that is, the maximum cut in the underlying undirected graph.

MAX2SAT (maximizing the number of satisfied clauses, where each clause has two literals) is also in **MAXSNP**. Here we have three input relations, G_0, G_1, and G_2. Intuitively, G_i contains all clauses with i negative literals; that is, $G_0(x, y)$ if and only if $(x \vee y)$ is a clause of the represented expression; $G_1(x, y)$ if and only if $(\neg x \vee y)$ is a clause; and $G_2(x, y)$ if and only if $(\neg x \vee \neg y)$ is a clause. With these somewhat complicated input conventions we can write MAX2SAT as $\max_{S \subseteq V} |\{(x, y) : \phi(G_0, G_1, G_2, S, x, y)\}|$, where ϕ is the following expression:

$$(G_0(x, y) \wedge (S(x) \vee S(y)) \vee (G_1(x, y) \wedge (\neg S(x) \vee S(y)) \vee (G_2(x, y) \wedge (\neg S(x) \vee \neg S(y)),$$

where S now stands for the set of **true** variables. It is not hard to see that the problem, as stated, indeed asks for the truth assignment that maximizes the

total number of satisfied clauses. A similar construction, with four relations, establishes that MAX3SAT is in **MAXSNP**.

For k-DEGREE INDEPENDENT SET, our input is a non-standard representation of a graph $G = (V, E)$ with maximum degree k in terms of a $(k+1)$-ary relation H. H contains the $|V|$ $(k+1)$-tuples $(x, y_1, \ldots, y_k)$ such that the y_i's are the neighbors of node x (with repetitions when x has fewer than k neighbors). The k-DEGREE INDEPENDENT SET problem can be written thus:

$$\max_{S \subseteq V} |\{(x, y_1, \ldots, y_k) : [(x, y_1, \ldots, y_k) \in H] \wedge [x \in S] \wedge [y_1 \notin S] \wedge \ldots \wedge [y_k \notin S]\}|.$$

S is the independent set.

Finally, k-DEGREE NODE COVER is in **MAXSNP** because it L-reduces to k-DEGREE INDEPENDENT SET (recall Example 13.1). Notice that, since $\mathbf{MAXSNP_0}$ contains by definition only maximization problems, an L-reduction to another problem in **MAXSNP** is the only way for a minimization problem, such as k-DEGREE NODE COVER, to be in **MAXSNP**. □

All four maximization problems in **MAXSNP** that we saw in Example 13.2 were shown in the previous section to have a polynomial-time ϵ-approximation algorithm, for some $\epsilon < 1$. This is no coincidence:

Theorem 13.8: Let A be a problem in $\mathbf{MAXSNP_0}$. Suppose that A is of the form $\max_S |\{(x_1, \ldots, x_k) : \phi\}|$. Then A has a $(1 - 2^{-k_\phi})$-approximation algorithm, where by k_ϕ we denote the number of atomic expressions in ϕ that involve S.

Proof: Consider an instance of A with universe V. For each k-tuple $v = (v_1, \ldots, v_k) \in V^k$ let us substitute these values for $x_1, \ldots, x_k$ in ϕ, to obtain an expression ϕ_v. There are three kinds of atomic expressions of ϕ_v, namely those that have relational symbols G_i (the input relations), $=$ (equality between the v_i's), and S. The former two kinds can be readily evaluated to **true** or **false** from the known values of the input relations and the v_i's, and substituted away from ϕ_v (this should be reminiscent of the similar construction in the proof of Theorem 5.9). It follows that ϕ_v is ultimately a Boolean combination of atomic expressions $S(v_{i_1}, \ldots, v_{i_r})$.

Thus, this instance of A is essentially a set of expressions of the form ϕ_v for all possible k-tuples v, and we are asked to assigning truth values to the various atomic expressions $S(v_{i_1}, \ldots, v_{i_r})$ (which we can consider as Boolean variables) so as to maximize the number of satisfied ϕ_v's. But this is an instance of MAXGSAT (the problem in which we are given a set of Boolean expressions, and we are asked to find the truth assignment that satisfies as many as possible, recall the Maximum Satisfiability subsection of Section 13.1). And the discussion preceding Theorem 13.2 shows how to approximate that problem with relative error at most $(1 - 2^{-m})$, where m is the number of Boolean variables appearing in each expression—in our case, k_ϕ. □

Thus, all optimization problems in **MAXSNP** share a positive approximability property (they all have some ϵ-approximation algorithm with $\epsilon < 1$, though not quite a polynomial-time approximation scheme), much the same way that all problems in **NP** share a positive algorithmic property —they can be solved in polynomial time by a nondeterministic algorithm, though not necessarily by a deterministic one. Whether all problems in **MAXSNP** have a polynomial-time approximation scheme is thus a most important question—the approximability counterpart of the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem that we have been seeking. Predictably, we shall now turn to identifying *complete problems.*

MAXSNP-Completeness

We say that a problem in **MAXSNP** is **MAXSNP**-complete if all problems in **MAXSNP** L-reduce to it. From the Corollary to Proposition 13.2 we have:

Proposition 13.3: If a **MAXSNP**-complete problem has a polynomial-time approximation scheme, then all problems in **MAXSNP** have one. □

Naturally, it is not *a priori* clear that **MAXSNP**-complete problems exist. But they do:

Theorem 13.9: MAX3SAT is **MAXSNP**-complete.

Proof: Since any problem in **MAXSNP** by defintion can be L-reduced to a problem in $\mathbf{MAXSNP_0}$, it suffices to show that all problems in $\mathbf{MAXSNP_0}$ can be L-reduced to MAX3SAT. Consider such a problem A defined by the expression $\max_S |\{(x_1, \ldots, x_k) : \phi\}|$. The proof of Theorem 13.8 essentially shows that A can be L-reduced to MAXGSAT. What we need to do is to further work on the Boolean expressions produced in that construction, so that we obtain an instance of MAX3SAT.

The expressions produced in the proof of Theorem 13.8 for each instance x of A are Boolean expressions of the form ϕ_v, with Boolean variables corresponding to whether the various tuples of constants of x belong to S. As usual, we can discard those expressions ϕ_v that are unsatisfiable. We can represent each of the remaining Boolean expressions ϕ_v as a Boolean circuit with $\wedge$, $\vee$, and $\neg$ gates. We can then use the construction employed in the reduction from CIRCUIT SAT to 3SAT (Example 8.3). That is, we replace each gate of the circuit by a set of two or three clauses stating that the relation between the input and output values are as specified by the sort of the gate (recall Example 8.3). We also add the clause (g), where g is the output gate. We repeat this for each satisfiable ϕ_v. The resulting set of all clauses thus created is the desired instance $R(x)$ of MAX3SAT. From any truth assignment T for this instance we immediately get a feasible solution $S = S(T)$ of the instance x of A, by simply recovering S from the Boolean values of the variables. The description of the L-reduction is complete.

But it remains to show that this is indeed an L-reduction. Each satisfiable expression ϕ_v is replaced by at most c_1 clauses, where c_1 is a constant depending on the size of ϕ, and therefore specific to A (it is essentially three times the number of Boolean connectives in ϕ). Thus, the m satisfiable expressions ϕ_v are replaced by at most $c_1 m$ clauses. The optimum value of instance x is at least some constant fraction of m, say $\text{OPT}(x) \geq c_2 m$ (for example, by Theorem 13.8, we can take $c_2 = 2^{-k_\phi}$). Since in $R(x)$ we can always set the Boolean variables so that all clauses except for those corresponding to the output gates are satisfied, $\text{OPT}(R(x)) \leq (c_1 - 1)m$, and the first condition on the definition of the L-reduction is satisfied with $\alpha = \frac{(c_1-1)}{c_2}$. It is easy to see that the second condition is satisfied, as usual, with $\beta = 1$, and the proof is complete. □

Amplifiers and Expanders

In order to reduce MAX3SAT to other important problems in **MAXSNP** (see Theorem 13.11 below), we need to establish the equivalent of Proposition 9.3, that is, that MAX3SAT remains **MAXSNP**-complete even if each variable appears at most three times in the clauses (we call this restricted problem 3-OCCURRENCE MAX3SAT). In Chapter 9 we proved this by replacing each occurrence of variable x by a new variable, say using variables $x_1, x_2, \ldots, x_k$, and then adding the clauses $(x_1 \Rightarrow x_2), (x_2 \Rightarrow x_3), \ldots, (x_k \Rightarrow x_1)$. This "cycle of implications" ensures that in any satisfying truth assignment all these new variables will have the same truth value.

This simple trick does not work in the present context. The "cycle of implications" construction *is not* an L-reduction from MAX3SAT to 3-OCCURRENCE-MAX3SAT. To see why, let us consider the (admittedly, somewhat far-fetched) instance y of MAX3SAT with clauses $(x), (x), \ldots, (x)$, $(\neg x), (\neg x), \ldots, (\neg x)$, where there are ℓ copies of (x) and ℓ of $(\neg x)$ (recall that we do not forbid fewer than three literals in a clause, or repetitions of clauses). Obviously, the optimum of y is $\text{OPT}(y) = \ell$.

If we perform the reduction sketched above and replace the 2ℓ occurrences of x by $x_1, \ldots, x_{2\ell}$ adding the clauses $(x_1 \Rightarrow x_2), (x_2 \Rightarrow x_3), \ldots, (x_{2\ell} \Rightarrow x_1)$, we obtain a new instance of MAX3SAT $R(y)$ with 4ℓ clauses. We would have liked to say that from any solution s of $R(y)$ we can recover a corresponding solution $S(s)$ of y. However, the optimum solution s of $R(y)$ satisfies all but one of the 4ℓ clauses as follows: $x_1, x_2, \ldots, x_\ell$ are **true**, and $x_{\ell+1}, x_{\ell+2}, \ldots, x_{2\ell}$ are **false**. Thus all clauses $(x_1), \ldots, (\neg x_{2\ell})$ are satisfied; furthermore all clauses of the form $(x_i \Rightarrow x_{i+1})$ are satisfied, *except for the clause* $(x_\ell \Rightarrow x_{\ell+1})$ —the only one with **true** assumption and **false** conclusion. Obviously, there is no meaningful solution $S(s)$ of y that we can recover from s.

Thus, our "cycle of implications" trick does not give us an L-reduction. Is there another, more sophisticated graph of implications that would do? Let us

start by identifying what is wrong with the cycle: There is a large subset of its nodes (namely, $x_1, x_2, \ldots, x_\ell$) that have a single edge leaving them. This is damaging, because it translates in the "cheating" truth assignment that satisfies all clauses except for the implication associated with this edge. What would have prevented such cheating is a directed graph where all sets of nodes have a "substantial" number of edges leaving them—equal to the number of nodes in the set. We formalize this below:

Definition 13.6: Consider a finite set X, and a directed graph $G = (V, E)$ where $X \subseteq V$. We assume that $|V| \leq c \cdot |X|$ where c is a constant, and that all nodes of G have either indegree one and outdegree two or vice-versa, except for nodes in X that have indegree and outdegree one. That is, $|E| \leq 2c \cdot |X|$.

We say that G is an *amplifier* for X if the following is true: For any subset $S \subseteq V$ containing $s \leq \frac{|X|}{2}$ nodes from X, $|E \cap S \times (V-S)|, |E \cap (V-S) \times S| \geq |S|$. That is, there are edges entering and leaving S that are no less than the number of X-nodes in S. □

Suppose that an amplifier G exists for any set X. Then we could use it to build a generalized "cycle of implications," and L-reduce MAX3SAT to 3-OCCURRENCE MAX3SAT, as follows: For each variable of the original formula, say with k occurrences, we connect the set of variables $X = \{x_1, \ldots, x_k\}$, together with up to ck new variables that make up all of V, by no more than $2ck$ implications *exactly as suggested by the edges of* G. We claim that the resulting construction is an L-reduction from MAX3SAT to 3-OCCURRENCE MAX3SAT.

In proof, notice first of all that if y is the original instance, with m clauses, then $R(y)$ has at most $(2c+1)m$ clauses, and thus the first condition is satisfied $\alpha = 2(2c+1)$ (we can satisfy at least half the clauses of y). More importantly, if s is any solution of $R(y)$, then we would be able to modify it so that, for each variable x of y, all k copies of x in $R(y)$, plus all additional nodes in V, have the same truth value —that of the majority of copies of x. This may result in $|S|$ copies of x changing values, and hence in our losing up to $|S|$ clauses satisfied in $R(y)$. However, because of the property of the amplifier, we know that we have *gained* at least $|S|$ clauses in the process (those implications that were going from the **true** to the **false** part of G). Hence, it is no loss of generality to assume that, in the optimum truth assignment of $R(y)$, all copies of each variable have the same truth value, and thus an equivalent truth assignment of y can be readily recovered. The distance from optimality is obviously the same in the two assignments.

We conclude that, if we assume that amplifiers exist, we have a proof that 3-OCCURRENCE MAX3SAT is **MAXSNP**-complete. We shall next show that amplifiers do exist. Once more, our argument will be non-constructive, in a way reminiscent of the proof of Theorem 11.6. We start by defining a related concept:

Definition 13.7: Let $\delta > 0$. A *δ-expander* for X is a graph $G = (X, E)$ where $|E| \leq c'|X|$ with the following property: For each subset $S \subseteq X$ of size s, $|E \cap S \times (V - S)| \geq \delta|S|$. □

From a δ-expander one can construct an amplifier. There are several problems to be fixed. First, there are only $\delta|S|$ edges leaving S instead of the required $|S|$; but this can be achieved by taking $\lceil \frac{1}{\delta} \rceil$ *copies of* G. Second, there is no guarantee that there are many edges *entering* $|S|$; this is easily taken care of *by adding all inverse edges, thus making the graph symmetric.* Finally, to enforce the indegree and outdegree constraints, for each node $x \in X$ with outdegree $k \geq 2$ we add to V $k - 1$ new nodes, that "fan out" the edges leaving x. A similar construction works for the indegrees. The resulting graph is an amplifier for X with $c = \frac{4c'}{\delta}$.

Thus, the following result establishes that amplifiers exist:

Lemma 13.2: For any $n \geq 2$, all sets X of size n have a δ-expander for some $\delta > 0$.

Proof: Consider a *random function F on X*, a random graph with all outdegrees one. That is, for each $x \in X$ we pick equiprobably and independently $F(x)$ to be any other node. Let S be a subset of X of size $|S| = s \leq \frac{n}{2}$. We call S *bad* if it has fewer than δs outgoing edges (where δ is a small number to be fixed later). What is the probability that S is bad?

To answer, delete for a moment from S the sources of the outgoing edges. The remaining subset T of size $|T| \geq (1 - \delta)|S|$ must be *introverted*, that is, for all nodes $x \in T$ we have $F(x) \in S$. For each node of T, the probability that it is mapped inside S is $\frac{s}{n}$, and thus the probability of T being introverted is at most $(\frac{s}{n})^{(1-\delta)|S|}$. The probability that an introverted subset T of S exists is therefore no more than this quantity, multiplied by the number of subsets of S of cardinality $(1 - \delta)s$, which is $\binom{s}{\delta s}$. Since we can approximate $\binom{n}{k}$ from above by $(\frac{en}{k})^k$ (see Problem 13.4.10), this latter number is at most $(\frac{e}{\delta})^{\delta s}$. We conclude that the probability that a set S of cardinality s is bad is no more than $(\frac{e}{\delta})^{\delta s}(\frac{s}{n})^{(1-\delta)s}$.

Suppose now that we take a graph F_k that is the union of k independently chosen random functions on X. Clearly, the probability that S is bad *now* is bounded from above by the kth power of this expression, $(\frac{e}{\delta})^{\delta ks}(\frac{s}{n})^{(1-\delta)ks}$. Finally, since there are at most $(\frac{en}{s})^s$ subsets of X of size s, the expected number of all bad subsets of size s in F_k is at most $(\frac{en}{s})(\frac{e}{\delta})^{\delta ks}(\frac{s}{n})^{(1-\delta)ks}$. Choosing $\delta = \frac{1}{10}$, $k = 8$, and recalling that $s \leq \frac{n}{2}$, after a calculation we conclude that the expected number of s-subsets of X with fewer than $\frac{s}{10}$ outgoing edges in F_8 is at most $(\frac{1}{2})^s$. Adding over all s's, we observe that the expected number of bad subsets in F_8 is *less than one.*

Thus, if we independently choose eight random functions on X and form F_8, the expected number of bad subsets is less than one. Clearly, there must

be one such experiment for which the outcome is zero bad subsets (otherwise the expectation would be at least one). We must conclude that *there is a graph with n nodes and outdegree eight such that all subsets S of size less than $\frac{n}{2}$ have at least $\frac{|S|}{10}$ outgoing edges.* This graph is a δ-expander with $\delta = \frac{1}{10}$ and $c' = 8$. □

Unfortunately, this result still does not give us the sought L-reduction from MAX3SAT to 3-OCCURRENCE MAX3SAT. The reason is that, although we know that amplifiers exist for all n and thus the construction is possible, we have not described an *algorithm* for generating an amplifier in space $\log n$ (in space $n \log n$, of course, we can try all possible graphs until we find our amplifier). The constructive form of Lemma 13.2 is much too involved to prove here (see the references in 13.4.11):

Lemma 13.2′: There is an algorithm which, given n in unary, generates in $\log n$ space an amplifier for a set of size n. □

From this important fact, our main result follows:

Theorem 13.10: 3-OCCURRENCE MAX3SAT is **MAXSNP**-complete. □

From Theorem 13.10 we can show several other **MAXSNP**-completeness results.

Theorem 13.11: The following problems are **MAXSNP**-complete:

(a) 4-DEGREE INDEPENDENT SET;
(b) 4-DEGREE NODE COVER;
(c) 5-OCCURRENCE MAX2SAT;
(d) MAX NAESAT;
(e) MAX-CUT.

Proof: For (a) we notice that the reduction in the proof of Theorem 9.4 is also an L-reduction from 3-OCCURRENCE MAX3SAT to 4-DEGREE INDEPENDENT SET. The properties of L-reduction are obvious, because the optimum value in the two instances is the same. That the maximum degree of the resulting graph is four follows from the fact that, since each variable has at most three occurrences, each literal occurrence has at most two contradicting literal occurrences. The only possible subtlety is that there may now be clauses with one or two literals; these are represented by single nodes or edges, respectively (as opposed to triangles).

Reducing 4-DEGREE INDEPENDENT SET to 4-DEGREE NODE COVER is trivial: The R part is the identity, and the S part produces $V - S$ from S (recall the Corollary 2 to Theorem 9.4). The reduction is an L-reduction (despite the fact that the same reduction between the unrestricted problems *is not* an L-reduction, recall Example 13.1), because both optimum values are lineraly related to the number of nodes.

The L-reduction from 4-DEGREE INDEPENDENT SET to 5-OCCURRENCE MAX2SAT, we have a variable for each node. For each node x we add to the instance of 5-OCCURRENCE MAX2SAT the clause (x), and for each edge $[x, y]$ we add the clause $(\neg x \vee \neg y)$. This completes the construction of the instance of 5-OCCURRENCE MAX2SAT. It is easy to see that the optimum truth assignment can be assumed to satisfy all edge clauses (because, if not, making one of the nodes **false** cannot decrease the number of satisfied clauses). Thus the optimum number of satisfied clauses equals the size of the maximum independent set plus $|E|$—and $|E|$ is a constant multiple of the maximum independent set, recall Example 13.1.

We can reduce MAX2SAT to MAX NAESAT (the maximization problem associated with the not-all-equal version of 3SAT) by the same reduction we used to show NAESAT **NP**-complete (Theorem 9.3): We add to each clause a new literal z. The optimum value in the two instances is the same, and so we have an L-reduction. Finally, the reduction from NAESAT to MAX-CUT in the proof of Theorem 9.5 is an L-reduction. □

13.3 NONAPPROXIMABILITY

We have been treating the question of whether **MAXSNP**-complete problems have polynomial-time approximation schemes very much like the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question. As it happens, the similarity runs a little deeper than anyone had expected: A sequence of deep and striking recent results has established that *the two questions are equivalent:* **MAXSNP**-complete problems have polynomial-time approximation schemes if and only if $\mathbf{P} = \mathbf{NP}$—naturally, this is the strongest negative result we could have hoped for approximability, short of proving that $\mathbf{P} \neq \mathbf{NP}$. Connecting the approximability issue with the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question required the development of a very interesting *alternative characterization of* **NP**, one that sees this class in a light much more appropriate for the discussion of the issue of approximability than our current view in terms of precise, rigid computations. We describe this fascinating result next.

Weak Verifiers

Let L be a language and M a Turing machine. We say that M *is a verifier for* L if L can be written as

$$L = \{x : (x, y) \in R \text{ for some } y\},$$

where R is a polynomially balanced relation decided by M. According to Proposition 9.1, a language L is in **NP** if and only if it has a polynomial-time verifier. As it turns out, the requirement that M be polynomial time can be relaxed:

Proposition 13.4 (Cook's Theorem, Weak Verifier Version): A language L is in **NP** if and only if it has a deterministic logarithmic-space verifier.

Proof: Since SAT is **NP**-complete, there is a reduction F from L to SAT. Define now $(x, y) \in R$ if and only if $y = F(x); z$, where z is a satisfying truth assignment of the Boolean expression $F(x)$. Obviously $x \in L$ if and only if there is a y such that $(x, y) \in R$. Furthermore, whether $(x, y) \in R$ can be decided in logarithmic space: The machine computing $F(x)$ compares one-by-one the bits of its output with y; telling whether z satisfies $F(x)$ in logarithmic space is trivial. □

So, the complexity of verifiers for **NP** can be reduced from polynomial-time to logarithmic-space. How much further can it be reduced? The important result of this subsection is a *surprisingly weak verifier for* **NP**.

Definition 13.8: Our new verifier is a *randomized* machine which makes use of *only* $\mathcal{O}(\log |x|)$ *random bits* when verifying whether $(x, y) \in R$. Furthermore, although the verifier has full access to the input x, *it has very limited access to the certificate* y: *It only examines a constant number of bits of* y. This is done as follows: On input x, and with random bit string $r \in \{0,1\}^{\lceil c \log |x| \rceil}$ for some $c > 0$, the verifier computes a *finite set of integers* $Q(x, r) = \{i_1, \ldots, i_k\}$, where k is a fixed integer and the i_j's are all at most $|y| = p(|x|)$. Then the i_jth symbol of y is found and written on a string of the verifier, $j = 1, \ldots, k$; the rest of the certificate y is not needed. Finally, the verifier performs a polynomial-time computation based on x, r, and $y_{i_1} \ldots y_{i_k}$. All computations of the verifier end with a "yes" or a "no".

We call such a machine a $(\log n, 1)$-*restricted verifier*, to record its two important resource restrictions: $\mathcal{O}(\log n)$ random bits, and a constant, $\mathcal{O}(1)$ that is, number of access steps. We say that a $(\log n, 1)$-restricted verifier decides a relation R if for each input x and alleged certificate y the following is true: If $(x, y) \in R$ then for all random strings the verifier ends up with "yes". If however $(x, y) \notin R$, then at least $\frac{1}{2}$ of the random strings cause the verifier to end up with "no". In other words, there can be a limited fraction of false positives, but no false negatives†. □

The novel characterization of **NP** alluded to above is this:

Theorem 13.12: A language L is in **NP** if and only if it has a $(\log n, 1)$-restricted verifier. □

---

† Recall that our goal here is to provide an alternative view of **NP**, one that will rid us of the "rigidity" of the conventional view, and thus create a framework appropriate for the study of approximability. In this regard randomness seems a most important ingredient, in that the possibility of false positives provides the necessary "imprecision" that was lacking in the conventional formulation of **NP**. This is reminiscent of the definition of $\mathbf{MAXSNP_0}$ (Definition 13.5), where instead of rigidly insisting that ϕ hold for all k-tuples, we settled for the set S that maximizes the number of k-tuples for which ϕ holds. It can be shown that the two maneuvers are in fact equivalent (see Problem 13.4.13).

The ingenious proof of Theorem 13.12 brings together many ideas and techniques that have been developed in recent years. For an account see 20.2.16 and 20.2.17.

Nonapproximability Results

Theorem 13.12 has some immediate and important consequences for the approximability of optimization problems:

Theorem 13.13: If there is a polynomial-time approximation scheme for MAX3SAT then **P = NP**.

Proof: Let $L \in \mathbf{NP}$, and let V be the $(\log n, 1)$-restricted verifier that decides the relation R as in Theorem 13.12, using $c \log n$ random bits and d accesses, for some positive constants c and d. It is not a loss of generality to assume that if $(x, y) \in R$ then $x; y \in \{0,1\}^*$ and $|x; y| = |x|^k$. Suppose now that there is a polynomial-time approximation scheme for MAX3SAT which achieves approximation $\epsilon > 0$ in time $p_\epsilon(n)$, a polynomial. Based on this, we shall describe a polynomial-time algorithm for deciding L.

Suppose that we wish to tell whether $x \in L$, where $|x| = n$. Let $r \in \{0,1\}^{c \log n}$, and consider the computation of V effected by this sequence of random choices. We wish to construct a Boolean expression that expresses the fact that the computation ends up in "yes". During this computation, V will seek access to d bits of y, say $y_{i_1(r)}, \ldots, y_{i_d(r)}$. Except for these bits of y, all other aspects of the computation of V or random choices r are completely determined. Thus, the outcome of the computation is a *Boolean function of these d bits considered as Boolean variables*, and thus can be expressed as a circuit C_r. We know (recall the proof of Proposition 4.3) that the number of gates of C_r is at most $K = 2^{2d}$, a constant. In turn, C_r can be expressed as a set of K or fewer clauses, denoted ϕ_r (recall the reduction from CIRCUIT SAT to SAT, Example 8.3). Notice that, no matter how we set the values of the variables $y_{i_1(r)}, \ldots, y_{i_d(r)}$, all but one of the clauses can be satisfied. Only certain settings of these variables can satisfy the last clause (the one stating that the outcome of the computation is "yes").

Repeating for all $2^{c \log n} = n^c$ possible sequences of random choices for V, we arrive at a set of at most Kn^c clauses, where the various groups of clauses share only the $y_{i_j(r)}$ variables. Because of the acceptance convention of our verifiers, we know the following: If $x \in L$, then there is a truth assignment (that is, a certificate y for x, together with the values of the gates of the various circuits C_r) that satisfies all clauses. And if $x \notin L$, any truth assignment must miss one clause in at least half of the groups; that is, at least a fraction of $\frac{1}{2K}$ of the clauses must be left unsatisfied.

This is where the assumed polynomial-time approximation scheme for MAX3SAT comes into play. We apply this scheme to the constructed set of

clauses with $\epsilon = \frac{1}{4K}$ —the time requirements will be $p_{\frac{1}{4K}}(Kn^c)$, a polynomial in n. If the scheme returns a truth assignment that satisfies more than a fraction of $1 - \frac{1}{2K}$ of the clauses, then we know that $x \in L$ (otherwise there is no truth assignment that satisfies more than such a fraction of the clauses). Otherwise, since the returned truth assignment is guaranteed to be within $\frac{1}{4K}$ of the optimum, we know that there is no truth assignment that satisfies all clauses, and thus $x \notin L$ (notice the similarity with the argument in the proof of Theorem 13.4 about the nonapproximability of the TSP). The proof is complete. □

Since MAX3SAT is in **MAXSNP**, we must conclude that *no* **MAXSNP**-*complete problem can have a polynomial-time approximation scheme*, unless of course **P** = **NP**:

Corollary 1: Unless **P** = **NP**, none of these problems have a polynomial-time approximation scheme: MAX3SAT, MAXNAESAT, MAX2SAT, 4-DEGREE INDEPENDENT SET, NODE COVER, and MAX-CUT. □

See the references for more results of this sort. For the unrestricted INDEPENDENT SET problem, we combine Theorem 13.13 with Theorem 13.6 to get something much more devastating:

Corollary 2: Unless **P** = **NP**, the approximation threshold of INDEPENDENT SET and CLIQUE is *one*. □

13.4 NOTES, REFERENCES, AND PROBLEMS

13.4.1 Problem: (a) Show that the greedy heuristic for NODE COVER (recall Section 13.1) never produces a solution which is more than $\ln n$ times the optimum.

(b) Find a family of graphs in which the $\ln n$ bound is achieved in the limit.

(c) NODE COVER is a special case of SET COVERING (recall Corollary 1 to Theorem 9.9). Why? Generalize the greedy heuristic to SET COVERING, and show that it has the same worst-case ratio.

These results are from

- D. S. Johnson "Approximation algorithms for combinatorial problems," *J.CSS, 9*, pp. 256–278, 1974,

which was the first systematic work on the subject of approximability of **NP**-complete problems. For the generalization in (c) see

- V. Chvátal "A greedy heuristic for the set cover problem," *Math. of Operations Research, 4*, pp. 233–235, 1979.

It has been shown that the approximability of SET COVERING is one (and in fact, unless $\mathbf{P} = \mathbf{NP}$, the best possible ratio is $\Theta(\log n)$), see

- C. Lund and M. Yannakakis "On the hardness of approximating minimization problems," *Proc. 25th ACM Symp. on the Theory of Computing*, pp. 286–295, 1993.

13.4.2 Problem: Recall the formal discussion of pseudopolynomial algorithms in Problem 9.5.31. Suppose that a strongly **NP**-complete optimization problem has the property that, for all inputs x, the optimum cost is at most $p(\text{NUM}(x))$ for some polynomial $p(n)$. (Notice that all problems we have seen so far in this book satisfy this property, with the intended meaning of $\text{NUM}(x)$.)

Show that such a problem has a fully polynomial-time approximation scheme if and only if $\mathbf{P} = \mathbf{NP}$.

13.4.3 Problem: Recall the BIN PACKING problem (Theorem 9.11); its minimization version seeks the smallest number of bins possible. Show that the approximation threshold of BIN PACKING is at least $\frac{1}{3}$. (Consider the problem of telling whether the number of bins needed is two or three, and recall the PARTITION problem shown **NP**-complete in Problem 9.5.33.)

13.4.4 Asymptotic approximation. There is something unsatisfying about the negative approximability result in the previous problem: It only holds for very small values of the number of bins. For all we know, there may be a polynomial-time heuristic for BIN PACKING that always comes within one bin of the optimum!

Obviously, this calls for a definition: We say that a heuristic M is an *asymptotic ϵ-approximation algorithm* if there is a constant $\delta > 0$ such that for all instances x

$$|c(M(x)) - \text{OPT}(x)| \leq \epsilon \cdot \max\{\text{OPT}(x), c(M(x))\} + \delta.$$

The *asymptotic approximation threshold* is again the greater lower bound for all ϵ's for which there is an asymptotic ϵ-approximation algorithm. If the asymptotic approximation threshold is zero, we say that the problem has an asymptotic polynomial-time approximation scheme.

(a) Consider an optimization problem A (maximization with goal K, or minimization with budget B), and the special case of problem A in which the goal (or budget) is bounded above by a constant c. We call this special case the *constant restriction* of A. Which of these problems have polynomial-time solvable constant restrictions, and which have an **NP**-complete constant restriction?

TSP, MINIMUM COLORING, MAX-CUT, BIN PACKING, KNAPSACK.

(b) Prove: If all constant restrictions of a problem are polynomial-time solvable, then its asymptotic approximation threshold coincides with the ordinary approximation threshold.

13.4.5 Problem: Problem 13.4.3 establishes that the approximation threshold for BIN PACKING is at least $\frac{1}{3}$, but tells us nothing about its asymptotic approximation threshold, arguably the more interesting quantity.

Consider then the following "first-fit" heuristic, where the n items are $a_1, \ldots, a_n$ and the capacity is C:

> Initialize n bins to empty $B[j] := 0, j = 1, \ldots, n$.
> For $i = 1, \ldots, n$ do:
> Find the smallest j with $B[j] + a_i \leq C$, and set $B[j] := B[j] + a_i$

(a) Show that the first-fit heuristic leaves at most one bin less than half-full.

(b) Conclude that the asymptotic approximation threshold of BIN PACKING is at most $\frac{1}{2}$.

13.4.6 Problem: But we can do much better; BIN PACKING is an example of a strongly **NP**-complete problem that has a polynomial-time asymptotic approximation scheme.

Fix any $\epsilon > 0$, and let $Q = \lfloor \epsilon C \rfloor$ be the "quantum size." Replace each item a_i by the quantity $a'_i = \lceil \frac{a_i}{Q} \rceil$. Notice that the value of each item must now be one of the "standardized" values $\{1, 2, \ldots, k\}$, where $k = \mathcal{O}(\frac{1}{\epsilon})$.

A *pattern* is a *sorted* sequence of positive integers adding up to k or less. For example, if $k = 4$, then $(1, 1, 2)$ and (3) are patterns. For $k = 4$ there are 12 different patterns: $\{(), (1), (1, 1), (1, 1, 1), (1, 1, 1, 1), (1, 1, 2), (1, 2), (1, 3), (2), (2, 2), (3), (4)\}$. In general, the number of patterns will be a fixed number P depending (exponentially) on k.

(a) Express the requirement that the items a'_i must be packed in m bins of capacity k as a set of equations in the nonnegative integer variables $x_1, \ldots, x_P$, where the intended meaning of x_j is *how many of the m bins are filled according to the jth pattern.*

(b) Using the fact that INTEGER PROGRAMMING with a constant number of variables can be solved in polynomial time (recall the discussion in 9.5.34), prove that there is an asymptotic polynomial-time approximation scheme for BIN PACKING.

For even better approximation algorithms for BIN PACKING see

- N. Karmarkar and R. M. Karp "An efficient approximation scheme for the one-dimensional bin-packing problem," *Proc. 23rd IEEE Symp. on the Foundations of Computer Science*, pp. 312–320, 1982.

13.4.7 Problem: Show that the MINIMUM UNDIRECTED KERNEL problem (recall Problem 9.5.10(g)) has approximation threshold one. (Modify the reduction used in its **NP**-completeness proof.)

13.4.8 Problem: Suppose that in a TSP instance the distances satisfy the *triangle inequality* $d_{ik} \leq d_{ij} + d_{jk}$. An approximation algorithm for this special case of the TSP is based on this idea: We find the *minimum spanning tree* of the cities (recall Problem 9.5.13). Taking each edge of the tree twice we get a graph that is connected, and has all degrees even. Such graphs are called *Eulerian*.

(a) Show that an Eulerian graph (possibly with multiple edges) has a cycle that visits each edge once (and each node possibly more than once).

(b) Show that in an instance of the TSP, if we have an Eulerial graph with total cost K, then we can find a tour with cost K or better.

(c) Show that there is a polynomial-time heuristic for the TSP with triangle inequality that yields a solution at most *twice* the optimum. (How does the minimum spanning tree compare with the optimum tour?)

There is a more sophisticated $\frac{2}{3}$-approximate heuristic, based on a *minimum matching of the odd-degree nodes* of the minimum spanning tree:

- N. Christofides "Worst-case analysis of a new heuristic for the traveling salesman problem," technical report GSIA, Carnegie-Mellon Univ., 1976.

This is the best heuristic known for the TSP with triangle inequality.

Suppose now that all distances in a TSP instance are either one or two. We wish to prove that this special case of the TSP is **MAXSNP**-complete. A first difficulty is this: Why is this special case of the TSP in **MAXSNP**?

It turns out that *any optimization problem with approximation threshold strictly less than one is in* **MAXSNP***!* In other words, **MAXSNP** is precisely the class of all optimization problems with approximation threshold strictly less than one (this is a result due to Madhu Sudan and Umesh Vazirani, communicated to me in 1993). To exemplify this for the TSP with distances one and two, we shall L-reduce it to MAXSAT. Given such an instance of the TSP with n cities, we know that its optimum value is between n and $2n$. Since telling whether the optimum of such an instance of the TSP is at most a given integer k is a problem in **NP**, for each k between n and $2n$ we can produce a Boolean expression ϕ_k such that (1) if the optimum is at most k c_k clauses of ϕ_k can be satisfied; and (2) otherwise all $c_k + 1$ clauses are satisfied.

Thus, the optimum of the instance of MAXSAT that combines all these clauses is $\sum_{k=n}^{2n} c_k + t$, where t is the optimum tour length.

(d) Show that this special case of the TSP is **MAXSNP**-complete.

When all distances are 1 or 2, the triangle inequality still holds (why?) and thus Christofides's heuristic still is still $\frac{1}{3}$-approximate. However, for the $1-2$ special case there is a polynomial $\frac{1}{6}$-approximate algorithm (and this is the best known):

- C. H. Papadimitriou and M. Yannakakis, "The traveling salesman problem with distances one and two," *Math. of Operations Research, 18*, 1, pp. 1–12, 1993.

Part (d) is also proved there

Incidentally, for the *asymmetric* generalization of the TSP, where d_{ij} is not necessarily equal to d_{ji}, but the triangle inequality still holds, no polynomial heuristic achieving a constant ratio is known.

13.4.9 Problem:

(a) Prove that the STEINER TREE problem is **MAXSNP**-complete even when all distances are 1 and 2.

(b) Consider the following heuristic for the STEINER TREE problem with triangle inequality: First, find the shortest distances between all nodes in the graph. Then, find the minimum spanning tree of the mandatory nodes, using as distances the shortest path lengths just found. Finally, build a Steiner tree by putting together all shortest paths used in the shortest spanning tree. Show that this algorithm never yields a Steiner tree with cost more than twice the optimum; thus the approximation threshold of STEINER TREE with triangle inequality is at most $\frac{1}{2}$.

The heuristic described above is from

- L. Kou, G. Markowsky, and L. Berman "A fast algorithm for Steiner trees," *Acta Informatica, 15*, pp. 141–145, 1981.

More recently, it was shown that the approximation threshold of STEINER TREE with triangle inequality is at most $\frac{5}{11}$:

- A. Z. Zelikowski "An $\frac{11}{6}$-approximation algorithm for the network Steiner problem," *Algorithmica, 9*, 5, pp. 463–470, 1993.

13.4.10 The doctoral dissertation

- V. Kann *On the Approximability of NP-complete Optimization Problems*, Royal Institute of Technology, Stockholm, Sweden, 1991

contains a comprehensive review of approximability results ca. 1991, as well as an extensive list of optimization problems and their approximability status.

13.4.11 Problem: Use Stirling's approximation formula $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ to show that $\binom{n}{k} \leq (\frac{en}{k})^k$.

13.4.12 The latest and simplest proof of Lemma 13.2′ can be found in

- O. Gabber and Z. Galil "Explicit construction of linear-sized superconcentrators," *J.CSS 22*, 407–420, 1981.

13.4.13 Problem: (a) Show that, unless **P** = **NP**, the approximation threshold of MINIMUM COLORING cannot be less than $\frac{1}{4}$. (Just recall that telling whether the minimum number of colors is 3 or 4 is **NP**-complete, Theorem 9.8.)

(b) Show that the *asymptotic* approximation threshold of MINIMUM COLORING cannot be less than $\frac{1}{4}$. (Replace each node with a large clique.)

(c) Can you further amplify the construction for (b) to show that, unless **P** = **NP**, the asymptotic approximation threshold of MINIMUM COLORING cannot be less than $\frac{1}{2}$?

Part (c) was proved in

- M. R. Garey and D. S. Johnson "The complexity of near-optimal graph coloring," *J.ACM, 23*, pp. 43–49, 1976.

It was the strongest negative result known about this important problem until it was proved in the paper by Lund and Yannakakis referenced in 13.4.1 that the approximation threshold for MINIMUM COLORING is *one*.

13.4.14 MAXSNP and weak verifiers. We could have defined the class of optimization problems **MAXSNP** not in terms of logical expressions, but in terms of $(\log n, 1)$-restricted verifiers (recall Theorem 13.12).

Let $k_1, k_2, k_3 > 0$, and let f be a polynomial-time computable function assigning to each string x and each bit string r with $|r| = k_2 \log |x|$ a set $f(x,r)$ of k_1 numbers in the range $1 \ldots |x|^{k_3}$, and let M be a polynomial-time Turing machine with three inputs. Define now this optimization problem:

"Given x, find the string y of length $|x|^{k_3}$ that maximizes the number of strings r with $|r| = k_2 \log |x|$ such that $M(x, r, y|_{f(x,r)}) =$ "yes"."

Let **MAXPCP**† be the class of all optimization problems of this form.

(a) Show that **MAXSNP** $\subseteq$ **MAXPCP**. (For each problem in $\mathbf{MAXSNP}_0$ defined in terms of an expression ϕ, function f computes, for each input relation G and values for the first-order variables, the finitely many positions in relation S that we must look in order to decide ϕ. What if the problem is not in $\mathbf{MAXSNP}_0$?)

(b) Show that **MAXPCP** $\subseteq$ **MAXSNP**. (The trick is to define the right input relation G for each f, M, and input x. G has $K = k_1 + k_2 + k_1 \cdot k_3$ arguments; we write $G(r_1, \ldots, r_{k_2}, b_1, \ldots, b_{k_1}, j_{11}, \ldots, j_{1k_3}, \ldots, j_{k_1 k_3})$. A K-tuple of integers in the range $0 \ldots |x| - 1$ are related by G if the following is true: (a) The ith element of $f(x,r)$ is

---

† **PCP** stands for "probabilistically checkable proofs." $\mathbf{PCP}(\log n, 1)$ was the name given in the paper referenced in 13.4.15 below to the class of all languages that have $(\log n, 1)$-restricted verifiers—that is to say, the class **NP**. The purpose of this problem is to point out the close relationship between this concept and **MAXSNP**.

the integer $j_{i1} \ldots j_{ik_3}$ in $|x|$-ary, where r is the bit string spelled by the r_j's in binary; and (b) If the bit of y that corresponds to the ith element of $f(x,r)$ equals b_i for all i, then $M(x,r,y|_{f(x,r)}) =$ "yes". On the other hand, relation S encodes y (say, it contains all k_3-tuples that correspond to 1-bits of y). Write a simple expression ϕ that says "$M(x,r,y|_{f(x,r)}) =$ "yes"".

13.4.15 The definition of **MAXSNP** and Theorems 13.8 through 13.11 are from

- C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *Proc. 20th ACM Symp. on the Theory of Computing*, pp. 229–234, 1988; also, *J.CSS* 1991.

13.4.16 Theorems 13.12 and 13.13 were proved in

- S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy "Proof verification and hardness of approximation problems," *Proc. 33rd IEEE Symp. on the Foundations of Computer Science*, pp. 14–23, 1992.

The proof of Theorem 13.12 is the culmination of several lines of investigation in complexity theory. Most immediately, it builds upon a slightly weaker result —namely that **NP** has $(\log n, (\log \log n)^k)$-restricted verifiers, as opposed to $(\log n, 1)$-restricted ones, enough to prove Corollary 2 of Theorem 13.13— which preceded it by a few weeks, and was presented in the same conference

- S. Arora, S. Säfra "Probabilistic checking of proofs," *Proc. 33rd IEEE Symp. on the Foundations of Computer Science*, pp. 2-13, 1992.

For an account of the developments that led to this remarkable result see Problems 20.2.16 and 20.2.17, as well as the following edition of the "NP-completeness column."

- D. S. Johnson "The tale of the second prover," *J. of Algorithms 13*, pp. 502–524, 1992.