بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

طراحی الگوریتم‌ها

حسین فلسفین

Handling NP-Hard Problems

In the absence of polynomial-time algorithms for problems known to be NP-hard, what can we do about solving such problems?

راهبرد اول:

The backtracking and branch-and-bound algorithms for these problems are all worst-case non-polynomial-time. However, they are often efficient for many large instances. Therefore, for a particular large instance of interest, a backtracking or branch-and-bound algorithm may suffice.

We try to design algorithms for solving hard problems and we accept their (worst case) exponential complexity if they are efficient and fast enough for most of the problem instances appearing in the specific applications considered.

If the actual inputs are small, an algorithm with exponential running time may be perfectly satisfactory.

The second approach: Approximation algorithms

An approximation algorithm for an NP-hard optimization problem is an algorithm that is not guaranteed to give optimal solutions, but rather yields solutions that are reasonably close to optimal. Often we can obtain a bound that gives a guarantee as to how close a solution is to being optimal.

It is a fascinating effect if one can jump from exponential complexity (a huge inevitable amount of physical work for inputs of a realistic size) to polynomial complexity (a tractable amount of physical work) due to a small change in the requirements — instead of an exact optimal solution one demands a solution whose cost differs from the cost of an optimal solution by at most $\varepsilon\%$ of the cost of an optimal solution for some $\varepsilon > 0$.

The third approach: Heuristics

The term heuristic in the area of combinatorial optimization is not unambiguously specified and so it is used with different meanings.

A heuristic algorithm in a very general sense is a consistent algorithm for an optimization problem that is based on some transparent (usually simple) strategy (idea) of searching in the set of all feasible solutions, and that does not guarantee finding any optimal solution. In this context people speak about local search heuristics, or a greedy heuristic, even when this heuristic technique results in an approximation algorithm.

In a narrow sense a heuristic is a technique providing a consistent algorithm for which nobody is able to prove that it provides feasible solutions of a reasonable quality in a reasonable (for instance, polynomial) time, but the idea of the heuristic seems to promise good behavior for typical instances of the optimization problem considered. Thus, a polynomial-time approximation algorithm cannot be considered as a heuristic in this sense, independently of the simplicity of its design idea.

Observe that the description of a heuristic in this narrow sense is a relative term because an algorithm can be considered to be a heuristic one while nobody is able to analyze its behavior. But after proving some reasonable bounds on its complexity and the quality of the produced solutions, this algorithm becomes an approximation algorithm and is not considered to be a heuristic any more.

Another general property of heuristics (independent of in which sense one specifies this term) is that they are very robust. This means that one can apply any heuristic technique to a large class of optimization problems, even if these problems have very different combinatorial structures. This is the main advantage of heuristics and the reason why they became so popular and widely used.

راهبرد چهارم (مهم):

Another approach is to find an algorithm that is efficient for a **subclass** *of instances of an NP-hard problem.*

We may be able to isolate important special cases that we can solve in polynomial time.

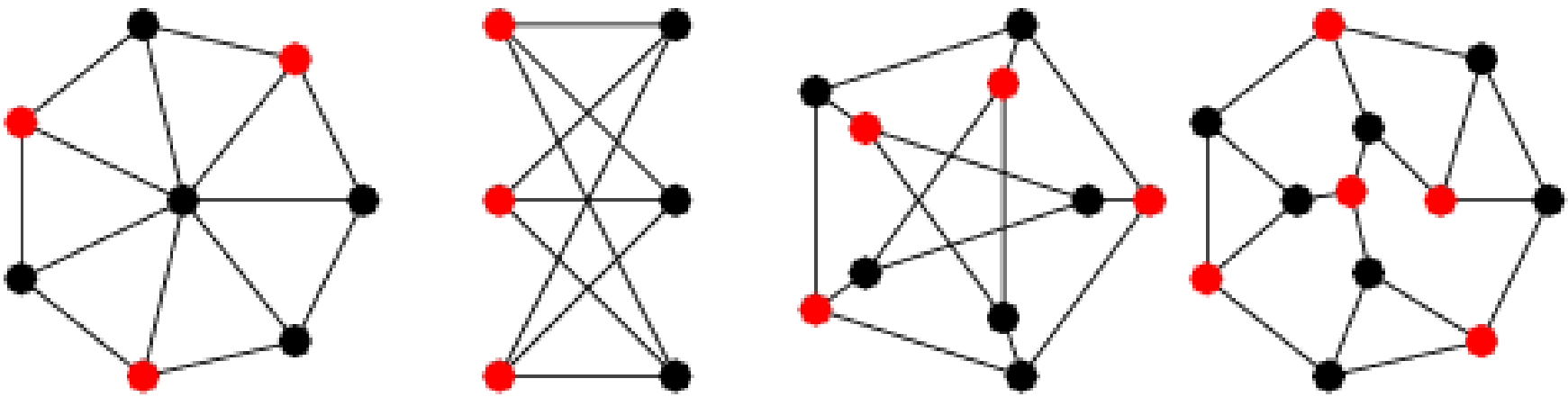
مثلاً:

It is well-known that 2-SAT is linear time solvable, while for every $k \geq 3$, k-SAT is NP-complete.

یادآوری: در یک نمونهٔ *2-SAT*، همهٔ کلاوزها دارای اندازهٔ ۲ هستند.

Maximum Independent Set Problem

A subset S of the vertex set V of a graph G is called independent if no two vertices of S are adjacent in G. $S \subseteq V$ is a maximum independent set of G if G has no independent set S' with $|S'| > |S|$.



مسئلهٔ مجموعهٔ مستقل بیشینه هم‌ارز با مسئلهٔ پوشش رأسی کمینه است:

Theorem: *A subset S of V is independent if and only if $V \setminus S$ is a covering of G.*

Proof. *S is independent if and only if no two vertices in S are adjacent in G. Hence, every edge of G must be incident to a vertex of $V \setminus S$. This is the case if and only if $V \setminus S$ is a covering of G.*

. . . . . .

Definition: *The number of vertices in a maximum independent set of G is called the independence number (or the stability number) of G and is denoted by $\alpha(G)$. The number of vertices in a minimum covering of G is the covering number of G and is denoted by $\beta(G)$.*

Proposition: ***For any graph*** G, $\alpha(G) + \beta(G) = n$.

Proof. ***Let*** S ***be a maximum independent set of*** G. $V \setminus S$ ***is a covering of*** G ***and therefore*** $|V \setminus S| = n - \alpha(G) \geq \beta(G)$. ***Similarly, let*** K ***be a minimum covering of*** G. ***Then*** $V \setminus K$ ***is independent and so*** $|V \setminus K| = n - \beta(G) \leq \alpha(G)$. ***These two inequalities together imply that*** $\alpha(G) + \beta(G) = n$.

. . . . . .

مسئلهٔ مجموعهٔ مستقل بیشینه یک مسئلهٔ بهینه‌سازی سخت است. به بیان دقیق‌تر:

Maximum independent set problem is a problem for which determining approximate solutions with constant bounded performance ratio is computationally intractable (it does not belong to APX). Thus, the maximum independent set problem is very hard to approximate.

For some special classes of graphs, it is possible to do better—in fact, much better. We observe that it is relatively easy to find the maximum independent set in a tree in polynomial time via a dynamic programming algorithm.

حال نشان می‌دهیم که برای درخت‌ها، مسئلهٔ مجموعهٔ مستقل بیشینه را می‌توان با بهره‌گیری از راهبرد برنامه‌ریزی پویا، در زمان چندجمله‌ای (خطی) حل کرد:

We suppose that the tree T is rooted at some node. The dynamic program will work in a bottom-up fashion, starting with the leaves and working up to the root.

Let T_u be the subtree of T rooted at a node u. The dynamic programming table will have two entries for each node u in the tree, $I(T_u, u)$ and $I(T_u, \varnothing)$:

* ***$I(T_u, u)$ is the size of a maximum independent set of T_u that includes u, and***

* ***$I(T_u, \varnothing)$ is the weight of the maximum independent set of T_u that excludes u.***

If u is a leaf, we can easily compute the two entries for u. Now suppose that u is an internal node, with k children $v_1, v_2, \ldots, v_k$, and suppose we have already computed the entries for each child. We can then compute the two entries for u.

حالت اول:

Clearly, if u is included in the independent set for T_u, then $v_1, v_2, \ldots, v_k$ must be excluded, so the maximum independent set for T_u including u is u plus the union of the maximum independent sets for the T_{v_i} excluding v_i; that is,

$$I(T_u, u) = 1 + \sum_{i=1}^{k} I(T_{v_i}, \varnothing).$$

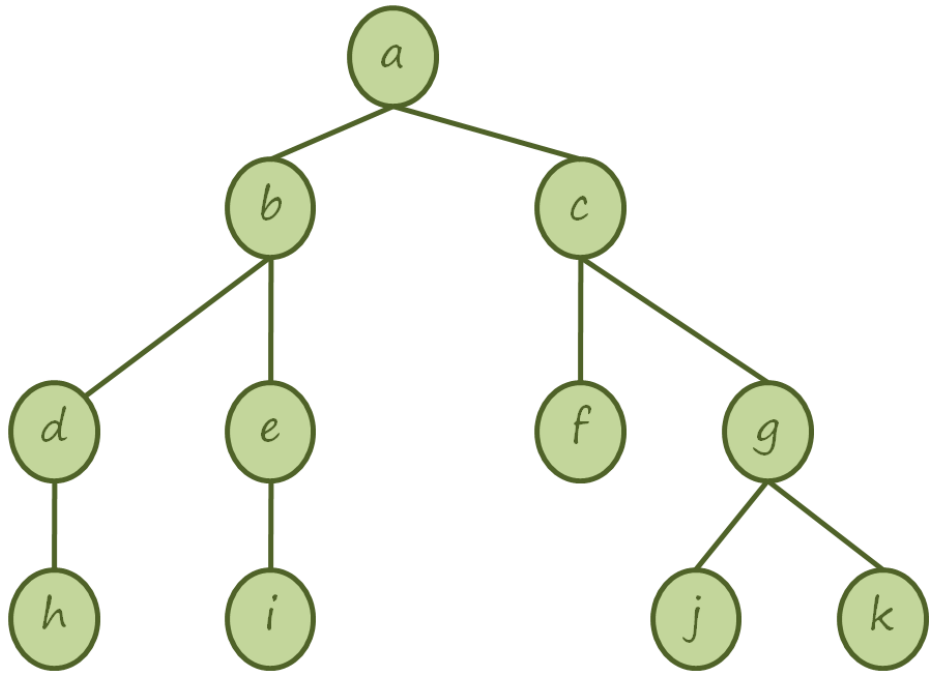
حالت دوم:

If u is excluded from the independent set for T_u, then we have the choice for each child v_i about whether we should take the maximum independent set of T_{v_i} including v_i or excluding v_i. Since there are no edges from any T_{v_i} to T_{v_j} for $i \neq j$, the decision for T_{v_i} can be made independently from that for T_{v_j} and we can choose either possibility; we simply pick the set of largest weight for each v_i, and this gives the maximum independent set for T_u excluding u. Thus,

$$I(T_u, \varnothing) = \sum_{i=1}^{k} \max(I(T_{v_i}, v_i), I(T_{v_i}, \varnothing)).$$

Once we have computed both entries for the root vertex r, we take the entry that has largest value, and this gives the maximum independent set for the entire tree.

مثال:



در جدول زیر، ستون‌های نظیر رئوسی که دارای عمق بیشتر هستند باید زودتر مقداردهی شوند:

	a	b	c	d	e	f	g	h	i	j	k
Include	6	3	3	1	1	1	1	1	1	1	1
Exclude	6	2	3	1	1	0	2	0	0	0	0

Trees are a very restricted class of graphs; we would like to be able to devise good algorithms for larger classes of graphs.

مثلاً، برای گراف‌های دوبخشی چطور؟ (تذکر: درخت‌ها خود گراف‌هایی دوبخشی هستند.) می‌توان از قضیهٔ معروف کُنیگ کمک گرفت:

König (1931): ***In a bipartite graph G, the minimum number of vertices that cover all the edges of G is equal to the maximum number of independent edges. In other words, the maximum cardinality of a matching in a bipartite graph is equal to the minimum cardinality of a vertex cover of its edges (i.e., $\beta(G)$).***

حال برای حل مسئلهٔ مجموعهٔ مستقل بیشینه برای گراف دوبخشی G، ابتدا باید به‌دنبال یک تطابق بیشینه بگردیم. با بهره‌گیری از الگوریتمی مانند الگوریتم «شکوفهٔ ادموندز»، این‌کار در زمان چندجمله‌ای میسّر است. حال قضیهٔ **König** بیان می‌دارد که اندازهٔ این تطابق، برابر با اندازهٔ پوشش رأسی کمینه است. از طرفی ما نشان دادیم که

$$\alpha(G) + \beta(G) = n.$$

پس برای به‌دست آوردن اندازهٔ مجموعهٔ مستقل بیشینه، یعنی $\alpha(G)$، کافی است تا اندازهٔ پوشش رأسی کمینه (یعنی همان اندازهٔ تطابق بیشینه) را از تعداد کل رئوس گراف کم کنیم. تمام!