Chapter 2

Branching Algorithms

> Nothing is particularly hard if you divide it into small jobs.
>
> *Henry Ford*

In this chapter, we present branching algorithms and various ways to analyze their running times. Unlike for other techniques to design exponential time algorithms (or polynomial time algorithms), like dynamic programming and inclusion-exclusion, it is far less obvious for branching algorithms how to obtain worst case running time bounds that are (close to) tight. The analysis of branching algorithms is a very important and non trivial factor in the design of branching algorithms; the design and the analysis of branching algorithms usually influence each other strongly and they go hand in hand. Also, branching algorithms usually perform faster on real life data and randomized instances than the (upper bound of the) worst case running time derived by the analysis: it is not uncommon that competitive SAT solvers in competitions like SAT Race and SAT Competition solve instances with thousands of variables, although no known algorithm for SAT has (worst case) time complexity $\mathcal{O}(1.9999^n)$.

Branching algorithms are recursive algorithms that solve a problem instance by reducing it to one or more "smaller" instances, solving these recursively, and combining the solutions of the subinstances to a solution for the original instance.

In the literature they appear with various names, for example Branch-and-Reduce, Branch-and-Bound, Branch-and-Search, Branching, Pruning the Search Tree, Backtracking, DPLL, or Splitting algorithms.

Typically, these algorithms

1. select a local configuration of the problem instance (selection),

2. determine the possible values this local configuration can take (inspection),

3. recursively solve subinstances based on these values (recursion), and
4. compute an optimal solution of the instance based on the optimal solutions of the subinstances (combination).

We call *reduction* a transformation (selection, inspection and the creation of the subinstances for the recursion) of the initial instance into one or more subinstances. We also call *simplification* a reduction to one subinstance, and *branching* or *splitting* a reduction to more than one subinstance.

Usually, the reduction and the combination steps take polynomial time and the reduction creates a constant number of subinstances (for exceptions, see for example [Ang05]). Polynomial time procedures to solve a problem for "simple" instances are viewed here as simplification rules, reducing the instance to the empty instance.

Let us illustrate branching algorithms by a simple algorithm for Maximum Independent Set. Consider Algorithm **mis** on the facing page. It contains two simplification rules. The first one (lines 1–2) solves Maximum Independent Set for graphs of maximum degree 2 in polynomial time. Clearly, for a collection of paths and cycles, the size of a maximum independent set can be computed in polynomial time: a maximum independent set of P_n has size $\lceil n/2 \rceil$ and a maximum independent set of C_n has size $\lfloor n/2 \rfloor$. The second simplification rule (lines 3–4) always includes vertices of degree 1 in the considered independent set. Its correction is based on the following observation.

Observation 2.1. *For a vertex v of degree 1, there exists a maximum independent set containing v.*

Proof. Suppose not, then all maximum independent sets of G contain v's neighbor u. But then we can select one maximum independent set, replace u by v in this independent set, resulting in an independent set of the same size and containing v — a contradiction. □

By the argument used in the proof of Observation 2.1, the algorithm is not guaranteed to go through all maximum independent sets of G, but is guaranteed to find at least one of them.

The first branching rule (lines 5–7) is invoked when G has at least two connected components. Clearly, the size of a maximum independent set of a graph is the sum of the sizes of the maximum independent sets of its connected components. If $V(G_1)$ (or $V \setminus V(G_1)$) has constant size, this branching rule may actually be viewed as a simplification rule, as G_1 (or $G \setminus V(G_1)$) is dealt with in constant time.

The second branching rule (lines 8–10) of the algorithm selects a vertex v of maximum degree; this vertex corresponds to the local configuration of the problem instance that is selected. Including or excluding this vertex from the

Algorithm **mis**(G)
Input : A graph $G = (V, E)$.
Output: The size of a maximum independent set of G.

```
1  if Δ(G) ≤ 2 then                          // G has maximum degree at most 2
2      return the size of a maximum independent set of G in polynomial time
3  else if ∃v ∈ V : d(v) = 1 then                                  // v has degree 1
4      return 1 + mis(G \ N[v])
5  else if G is not connected then
6      Let G_1 be a connected component of G
7      return mis(G_1) + mis(G \ V(G_1))
8  else
9      Select v ∈ V such that d(v) = Δ(G)          // v has maximum degree
10     return max(1 + mis(G \ N[v]), mis(G \ v))
```

Figure 2.1: Algorithm **mis**(G), computing the size of a maximum independent set of any input graph G

current independent set are the values that this local configuration can take. The subinstances that are solved recursively are $G \setminus N[v]$ — including the vertex v in the independent set, which prevents all its neighbors to be included — and $G \setminus v$ — excluding v from the independent set. Finally, the computation of the maximum in the last line of the algorithm corresponds to the combination step.

2.1 Simple Analysis

To derive upper bounds for the running time of a branching algorithm, let us describe its behavior by a model which consists of a set of univariate constraints.

Definition 2.2. Given an algorithm A and an instance I, $T_A(I)$ denotes the running time of A on instance I.

Lemma 2.3. *Let A be an algorithm for a problem P, and $\alpha > 0$, $c \geq 0$ be constants such that for any input instance I, A reduces I to instances $I_1, \ldots, I_k$, solves these recursively, and combines their solutions to solve I, using time $\mathcal{O}(|I|^c)$ for the reduction and combination steps (but not the recursive solves) and such that for any reduction done by Algorithm A,*

$$(\forall i : 1 \leq i \leq k) \quad |I_i| \leq |I| - 1, \text{ and} \tag{2.1}$$

$$2^{\alpha \cdot |I_1|} + \cdots + 2^{\alpha \cdot |I_k|} \leq 2^{\alpha \cdot |I|}. \tag{2.2}$$

Then A solves any instance I in time $\mathcal{O}(|I|^{c+1})2^{\alpha \cdot |I|}$.

Proof. The result follows easily by induction on $|I|$. For the base case, we assume that the algorithm returns the solution to an empty instance in time $\mathcal{O}(1)$. Suppose the lemma holds for all instances of size at most $|I| - 1$, then

$$\begin{aligned}
T_A(I) &= \mathcal{O}(|I|^c) + \sum_{i=1}^{k} T_A(I_i) && \text{(by definition)}\\
&= \mathcal{O}(|I|^c) + \sum \mathcal{O}(|I_i|^{c+1}) 2^{\alpha\cdot|I_i|} && \text{(by the inductive hypothesis)}\\
&= \mathcal{O}(|I|^c) + \mathcal{O}((|I|-1)^{c+1}) \sum 2^{\alpha\cdot|I_i|} && \text{(by (2.1))}\\
&= \mathcal{O}(|I|^c) + \mathcal{O}((|I|-1)^{c+1}) 2^{\alpha\cdot|I|} && \text{(by (2.2))}\\
&= \mathcal{O}(|I|^{c+1}) 2^{\alpha\cdot|I|}.
\end{aligned}$$

The final equality uses that $\alpha \cdot |I| > 0$ and holds for any $c \geq 0$. □

Let us use this lemma to derive a vertex-exponential upper bound of the running time of Algorithm **mis**, executed on a graph $G = (V, E)$ on n vertices. For this purpose we set $|G| := |V| = n$. We may at all times assume that n is not a constant, otherwise the algorithm takes constant time.

Determining if G has maximum degree 2 can clearly be done in time $\mathcal{O}(n)$. By a simple depth–first–search, and using a pointer to the first unexplored vertex, the size of a maximum independent set for graphs of maximum degree 2 can also be computed in time $\mathcal{O}(n)$. Checking if G has more than one connected component can be done in time $\mathcal{O}(n + m) = \mathcal{O}(n^2)$. Finding a vertex of maximum degree and the creation of the two subinstances takes time $\mathcal{O}(n+m) = \mathcal{O}(n^2)$. Addition and the computation of the maximum of two numbers takes time $\mathcal{O}(1)$.

For the first branching rule, we obtain a set of constraints for each possible size s of $V(G_1)$:

$$(\forall s : 1 \leq s \leq n-1) \quad 2^{\alpha\cdot s} + 2^{\alpha\cdot(n-s)} \leq 2^{\alpha\cdot n}. \tag{2.3}$$

By convexity of the function 2^x, these constraints are always satisfied, irrespective of the value of $\alpha > 0$ and can thus be ignored. Here we suppose that n is not a constant, otherwise the algorithm takes only constant time.

For the second branching rule, we obtain a constraint for each vertex degree $d \geq 3$:

$$(\forall d : 3 \leq d \leq n-1) \quad 2^{\alpha\cdot(n-1)} + 2^{\alpha\cdot(n-1-d)} \leq 2^{\alpha n}. \tag{2.4}$$

Dividing all these terms by $2^{\alpha n}$, the constraints become

$$2^{-\alpha} + 2^{\alpha\cdot(-1-d)} \leq 1. \tag{2.5}$$

Then, by standard techniques [Kul99], the minimum α satisfying all these constraints is obtained by setting $x := 2^\alpha$, computing the unique positive real root of each of the characteristic polynomials

$$c_d(x) := x^{-1} + x^{-1-d} - 1,$$

by Newton's method, for example, and taking the maximum of these roots. Alternatively, one could also solve a mathematical program minimizing α subject to the constraints in (2.5) (the constraint for $d = 3$ is sufficient as all other constraints are weaker). The maximum of these roots (see Table 2.1) is obtained for $d = 3$ and its value is $1.380277\ldots \approx 2^{0.464958\ldots}$.

d	x	α
3	1.3803	0.4650
4	1.3248	0.4057
5	1.2852	0.3620
6	1.2555	0.3282
7	1.2321	0.3011

Table 2.1: Positive real roots of $c_d(x)$

Applying Lemma 2.3 with $c = 2$ and $\alpha = 0.464959$, we find that the running time of Algorithm **mis** is upper bounded by $\mathcal{O}(n^3) \cdot 2^{0.464959 \cdot n} = \mathcal{O}(2^{0.4650 \cdot n})$ or $\mathcal{O}(1.3803^n)$. Here the $\mathcal{O}$ notation permits to exclude the polynomial factor by rounding the exponential factor.

2.2 Lower Bounds on the Running Time of an Algorithm

One peculiarity of branching algorithms is that it is usually not possible by the currently available running time analysis techniques to match the derived upper bound of the running time by a problem instance for which the algorithm really takes this time to compute a solution. Exceptions are brute-force branching algorithms that go through all the search space or algorithms enumerating objects for which tight bounds on their number are known, like the enumeration of maximal independent sets [MM65] or maximal induced bicliques [GKL08].

Lower bounds on the running time of a specific algorithm are helpful as they might give indications which instances are hard to solve by the algorithm, an information that might suggest attempts to improve the algorithm. Moreover, the design (or its attempt) of lower bound instances might give indications on

which "bad" structures do not exist in an instance unless some other "good" structures arise during the execution of the algorithm, which might hint at the possibility of a better running time analysis. Finally, it is desirable to sandwich the true worst case running time of the algorithm between an upper and a lower bound to obtain more knowledge on it as a part of the analysis of the algorithm.

Lower bounds are usually obtained by describing an infinite family of graphs for which the behavior of the algorithm is "cyclic", that is it branches on a finite number of structures in the instance in a periodic way.

To derive a lower bound of the running time of Algorithm **mis**, consider the graph $G = P_n^2$, depicted in Figure 2.2 — the second power of a path on n vertices, obtained from a path P_n on n vertices by adding edges between every two vertices at distance at most two in P_n. Suppose $n \geq 9$, then none of the simplification rules applies to G. The algorithm selects some vertex of degree 4; here, we — the designers of the lower bound — have the freedom to make the algorithm choose a specific vertex of degree 4, as it does not itself give any preference. Suppose therefore, that it selects v_3 to branch on. It creates two subproblems:

- $G \setminus N[v_3]$, a graph isomorphic to P_{n-5}^2, and
- $G \setminus v_3$, a graph isomorphic to P_2 connected with one edge to the first vertex of a P_{n-3}^2. In the first recursive step of **mis**, the reduction rule on vertices of degree at most 1 includes v_1 in the independent set and recurses on the graph isomorphic to P_{n-3}^2.

Now, on each of these subproblems of sizes $n-3$ and $n-5$, the behavior of the algorithm is again the same as for the original problem of size n. Therefore, the running time of the algorithm can be lower bounded by $\Omega(x^n)$ where x is the positive root of

$$x^{-3} + x^{-5} - 1,$$

which is $1.193859\ldots \approx 2^{0.255632\ldots}$. This gives a lower bound on the running time of Algorithm **mis** of $\Omega(1.1938^n)$ or $\Omega(2^{0.2556 \cdot n})$.

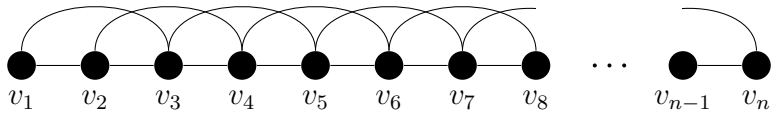

Figure 2.2: Graph P_n^2 used to lower bound the running time of Algorithm **mis**

Let us take a closer look at the decisions made by the algorithm with a P_n^2 as input. The size of the independent set increases by 1 when including v_3 and also by 1 when excluding v_3 and including v_1 by a simplification rule. But the instance

obtained by the second recursive call is, after the application of the simplification rule, a subgraph of the instance of the first recursive call. Therefore, the second recursive call always leads to a solution that is at least as good as the one obtained in the first recursive call. This is a special case of a set of local configurations where there exist two vertices u, v such that $N[u] \subseteq N[v]$. In such a situation, the algorithm can just exclude v from being in the maximum independent set it computes: consider a maximum independent set I_v of G containing v, then $I_v \setminus \{v\} \cup \{u\}$ is a maximum independent set of G not containing v.

Thus, we could enhance the algorithm by adding the corresponding simplification rule; see Figure 2.3 [1].

if $\exists u, v \in V : N[u] \subseteq N[v]$ **then**
  return **mis**$(G \setminus v)$

Figure 2.3: Additional simplification rule for Algorithm **mis**

2.3 Measure Based Analysis

One drawback of the analysis presented in Section 2.1 is that, when reducing the problem instance to several subinstances, many structural changes in the instance are not accounted for by a simple analysis in just the instance size. Therefore, let us in this section use a potential–function method akin to the measures used by Kullmann [Kul99], the quasiconvex analysis of Eppstein [Epp06], the "Measure and Conquer" approach of Fomin et al. [FGK05a], the (dual to the) linear programming approach of Scott and Sorkin [SS07a], and much older potential–function analyses in mathematics and physics.

For Algorithm **mis**, for example, the simple analysis does not take into account the decrease of the degrees of the neighbors when deleting a vertex from the graph. Taking this decrease of the degrees into account is particularly useful when deleting a vertex adjacent to one or more vertices of degree 2: their degree decreases to 1 and they (as well as their neighbors) are removed by a simplification rule.

Therefore, let us in a first step model the worst case running time of the algorithm by a set of multivariate constraints, where the variables correspond to the structures whose changes we would like to take into account. These structures may depend as well on the input instance the algorithm is currently considering, as on the output it is currently generating. Examples of parameters that the analysis may rely on are the number of vertices/variables of certain degrees, the

[1] The simplification rule for vertices of degree 1 becomes obsolete by adding this rule as vertices of degree 1 always fall under the scope of the new rule

Thus, we obtain the following recurrence where the maximum ranges over all $d \geq 3$, all $p_i, 2 \leq i \leq d$ such that $\sum_{i=2}^{d} p_i = d$ and all k such that $2 \leq k \leq d$:

$$T\left(\{n_i\}_{i\geq 1}\right) = \max_{d,p_2,\ldots,p_d,k} \begin{cases} T\left(\left\{n_i - p_i + p_{i+1} - \mathsf{K}_\delta(d=i)\right\}_{i\geq 1}\right) \\ +T\left(\left\{n_i - p_i - \mathsf{K}_\delta(d=i) - \mathsf{K}_\delta(k=i) + \mathsf{K}_\delta(k=i+1)\right\}_{i\geq 1}\right) \end{cases} \tag{2.6}$$

Here, $\mathsf{K}_\delta(\cdot)$ is the logical Kronecker delta [CB94], returning 1 if its argument is true and 0 otherwise.

Remark 1. This model of the worst case running time of the algorithm makes the assumption that it is always beneficial to decrease the degree of a vertex. When $N[v]$ is deleted, many more vertices in $N^2(v)$ could have their degree reduced (also by more than 1). Such assumptions are often necessary to limit the number of cases that need to be considered in the analysis.

In order to make the number of terms in recurrence (2.6) finite, let us restrict the running time analysis to graphs of maximum degree 5 in this section. In Section 2.7, we will combine an analysis for graphs of maximum degree 5 with the simple analysis of the previous section to derive an overall running time for Algorithm **mis** for graphs of arbitrary degrees.

Based on the multivariate recurrence (2.6) for $3 < d < 5$, we would now like to compute an upper bound of the algorithm's running time. Eppstein [Epp04, Epp06] transforms the models based on multivariate recurrences into models based on weighted univariate linear recurrences and shows that there exists a set of weights for which the solution of one model is within a polynomial factor of the solution of the other model.

Definition 2.4. A *measure* μ for a problem P is a function from the set of all instances for P to the set of non negative reals.

To analyze Algorithm **mis**, let us use the following measure of a graph G of maximum degree 5, which is obtained by associating a weight to each parameter $\{n_i\}_{1\leq i\leq 5}$:

$$\mu(G) := \sum_{i=1}^{5} w_i n_i, \tag{2.7}$$

where $w_i \in \mathbb{R}^+$ for $i \in \{1, \ldots, 5\}$ are the weights associated with vertices of different degrees.

With this measure and the tightness result of Eppstein, we may transform recurrence (2.6) into a univariate recurrence. For convenience, let

$$(\forall d : 2 \leq d \leq 5) \quad h_d := \min_{2\leq i\leq d} \{w_i - w_{i-1}\}, \tag{2.8}$$