

Hill climbing is a mathematical optimization algorithm. Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the algorithm takes an arbitrary $\vec{x} \in \mathcal{D}_f$ and compares $f(\vec{x})$, $f(\Delta\vec{x})$. If the shift increases the function, it lets $\vec{x} := \Delta\vec{x}$; otherwise it lets $\vec{x} := -\Delta\vec{x}$. The process is repeated until some criteria is met.

Hill climbing can be used to optimize the greedy coloring of a graph. The use of this algorithm for greedy coloring is justified by a nice theorem.

Given a graph $G = (V, E)$, the parameter of the Greedy coloring algorithm \mathcal{G} is an order $O = v_{i_1}, \dots, v_{i_n}$ of the vertices, where the i_j define the coloring order of $V = \{v_1, \dots, v_n\}$. Thus, shifts in the parameter space correspond to changes in the coloring order.

Such permutations lack a nice property of real functions; namely, that their parameters can only be shifted in two directions: positive or negative. Instead of these two choices, an order O of n vertices has $n!$ permutations. We cannot try them out and chose the one that maximizes our function.

However, there is a property which points to permutations that may not necessarily improve the coloring, but are guaranteed not to worsen it. Such property reduces drastically the space of candidate permutations. In fact, given an initial ordering O , this space of alternative permutations is often so small that none of them improve the coloring. Thus, it is common to generate k random initial orderings O_1, \dots, O_k , so as to explore their (reduced) permutation spaces, in hope that one of their permutations will in fact improve the coloring.

Theorem. Let $G = (V, E)$ a graph with a proper coloring of r colors $C = \{c_1, \dots, c_r\}$. Let $V_{c_i} := \{x \in V : c(x) = c_i\}$. Let $P : C \rightarrow C$ a permutation of c_1, \dots, c_r ; i.e. $P(c_i) = c_j$ entails the i th color becomes the j th color.

\mathcal{G} with the order $V_{P(c_1)}, \dots, V_{P(c_r)}$ colors G with at most r colors.

The proof of this theorem is simple to do inductively over r . If $r = 1$ the case is trivial. Assume the theorem holds for $r = k$. Let $x_0 \in \bigcup_{i=0}^{k+1} V_{P(c_i)}$. If x_0 belongs to any of the first k sets, \mathcal{G} colors it with one out of k colors by hypothesis. If $x_0 \in V_{P(c_{k+1})}$, the case where \mathcal{G} colors it with a color less than $k + 2$ makes our statement true. Assume this is not the case. Then there is some $y_0 \in \Gamma(x_0)$ such that $c(y_0) = k + 1$. But then $y_0 \in V_{P(c_{k+1})}$. And since x_0 is also in this set, we should have $c(x_0) = c(y_0)$. Then the coloring is not proper. (\perp)

Informally, if \mathcal{G} colored V into groups V_{c_1}, \dots, V_{c_r} , then coloring first the vertices with color $P(c_1), P(c_2), \dots, P(c_r)$ is at least as good as the original coloring. The permutation P that we must use is arbitrary: the theorem states nothing special about it. In general, one uses permutations that put vertices with highest colors or

highest degrees first, since these are the problematic ones. We will define a few such permutations.

Permutation 0. The first permutation will order the vertices from last to first color. Thus, assuming \mathcal{G} used r colors, the order will be V_{c_r}, \dots, V_{c_1} .

A pseudo-code implementation:

```

 $O = \text{int Array}[n]$ 
 $\mathcal{D} := \text{Queue Array}[r]$ 
for  $1 \leq i \leq n$  do
     $\text{push!}(i, \mathcal{D}[c(v_i)])$ 
end
int  $i := 0$ 
for  $queue$  in  $\mathcal{D}$  do
    while  $queue \neq \emptyset$  do
         $O[i] = \text{pop}(queue)$ 
         $i = i + 1$ 
    end
end
return  $O$ 

```

The first for loop is $O(n)$. Since $\sum_i^r |\mathcal{D}_i| = n$, the second for loop with its inner while is $O(n)$. \therefore The algorithm is $O(n)$.

Permutation 1. The second permutation we will write is a simple cardinality permutation. The color with the most number of vertices goes first, the color with the least number of vertices goes last. There are at most $O(n)$ sets V_{c_i} . Using QSort we have $O(n \log n)$ complexity.

Permutation 2. We will order the colors using their divisibility. First will come all colors divisible by four, then all colors divisible by 2, and then all other colors. In pseudo-code,

```

 $x = |\{c \in C : c \equiv 0 \pmod{4}\}|$ 
 $y = |\{c \in C : c \equiv 0 \pmod{2} \wedge c \not\equiv 0 \pmod{4}\}|$ 
 $O = \text{int Array}[n]$ 
int  $u = v = w = 0$ 
int  $index$ 
for  $i \in \{1, \dots, r\}$  do
    while  $\mathcal{D}[i] \neq \emptyset$  do
         $v = pop(\mathcal{D}[i])$ 
        if  $i \equiv 0 \pmod{4}$  do
             $index = u$ 
             $u = u + 1$ 
        else if  $i \equiv 0 \pmod{2}$  do
             $index = x + v$ 
             $v = v + 1$ 
        else do
             $index = x + y + w$ 
             $w = w + 1$ 
        fi
         $O[index] = v$ 
    od
od

```

which is $O(n)$ of course.

I generated a random graph of $n = 30$ vertices and $m = 32$ edges. I tested the permutations and found that the last permutation coloured the graph with 1 less color than the other ones. To the left, the Greedy coloring over the natural order of the vertices; to the right, the coloring over the order of **Permutation 2**.