# On the Jaggedness of Discrete Spaces and the Effectiveness of Local Search

June 2, 2019

## 1 Problem Description

On many occasions, I have found it necessary to find the max or the argmax over a function defined over some discrete space. Most often, it's a function that takes as input some binary string of fixed length, and outputs a number, although in principle it could be any discrete-valued function. In this document, I describe my thoughts about optimizing these kinds of functions.

Typically, the way people optimize functions like these is to do some kind of local search, meaning that you go from one vertex of the graph that the function is defined over to one of its neighbors, hoping to improve upon the function result, until you reach a local optimum. For example, if you were trying to find the maximum of the function $f(a, b) = a + b$, where $a$ and $b$ can both be either 0 or 1, you might start at $(0, 0)$ (whose corresponding function value is 0), find that you can improve upon it by evaluating at neighbor $(0, 1)$ (whose corresponding function value is 1), and then in the next step find a local (and, as it happens, global) maximum at $(1, 1)$ (with a corresponding function value of 2). How exactly you do decide to do this is a question of details; for example, you can try, at each step, searching every neighbor and moving to the neighbor with the highest value (which I'll call "deterministic neighbor search"). Maybe the most prominent algorithm of this kind that I know about is Markov Chain Monte Carlo; my personal preference for an algorithm like this one is to try the neighbors in a random order, and as soon as you find one that is an improvement, to move to it[1].

[1] My vague understanding is that this algorithm is equivalent to MCMC with some parameter or other set to infinity; I've found this to be the algorithm that generally finds local optima in a minimum of function evaluations, and it has the added benefit of simplicity.

In any case, my hope is that none of this is too important, and that the stuff I say in this document can remain general in terms of what search algorithm you end up deciding to use. The question of which search algorithm is best for what situation is an interesting one, but not one that I'm going to explore much here. Suffice it to say that the situation of interest is that we have some discrete-valued function $f(\vec{x})$ that we want to optimize, and some search algorithm $S(f, \vec{x}_0)$ that, given $f$ and an initial point $x_0$, finds a local optimum of the function $x^*$.

Taking $g$ as a given, it's an interesting question to ask which functions $f$ lend themselves best to optimization, and how we might efficiently measure this property. Assuming it's straightforward to find a local optimum of $f$ using $S$—a reasonable assumption, for halfway reasonable functions $f$ and sensible choices of $S$—how easy it is to globally optimize $f$ comes down to how good the local optima of $f$ are, relative to its global optimum. For most functions $f$ we use in practice, this will directly relate to how many local optima there are. The more local optima $f$ has, the less special each one is; we will need to perform more local searches from more different places before we can expect to have found the function's global optimum. I call functions with many local optima "jagged," and spaces with few local optima "smooth." I also sometimes use the term "peaks" to refer to local optima.

Intuitively, we will have an easier time find the global optimum (or values near the global optimum) for functions that are smooth than for functions that are jagged. For instance, imagine we are trying to find the global optimum of a function $f$ of $n$ binary variables (so the function has $2^n$ vertices). Imagine that $f$ has $m$ peaks, and that peaks of the function can be found using a local search algorithm $S$ in $O(n)$ time (which is true for most functions $f$). There are some assumptions buried here which I'll explore eventually—chief among them that the peaks behave "alike" and that the global optimum behaves typically for a peak—but in this fairly typical situation, it is reasonable to assume that if we try repeatedly running local searches from random points in the space, we'll find the global optimum in $O(mn)$ time. Note here that our runtime scales with the number of peaks $m$, matching our intuition that functions with many peaks (*jagged* functions) are harder to globally optimize.

Suppose that our space is too large and jagged for us to reasonably expect to be able to find the function's global maximum, but we'd still like to find the best$^2$vertex we can in the time we have available. If all we want is to find a vertex that's the $k$-largest in the graph or better, taking the same assumptions as before, it will take us $\Theta(mn/k)$ time before we're likely to

have found such a vertex.

## 2 Estimating the Jaggedness of a Space

If we accept the analysis near the end of the previous section, it is necessary to know how many peaks $m$ a function has before one can infer how good the best vertex we found is, having searched for some fixed amount of time. We might therefore hope for an efficient algorithm for estimating how many peaks a function has. That's the topic of this section.

I call a vertex $\vec{x}$ a *satellite* of a peak $\vec{p}$ if a local search initialized at $\vec{x}$ terminates at $\vec{p}$—in other words, if $S(f, \vec{x}) = \vec{p}$. (Naturally, the notion of being a satellite here only makes sense given the context of a specific $S$ and $f$.)

[2]When we talk about how "good" a vertex is, we generally won't be talking about its value in an absolute sense, but rather how large its value is relative to that of the other vertices in the graph, since that's all that many of the search algorithms we might use care about. For example, the graph's global optimum is the global optimum because it has the largest value of any vertex, but we might also be interested in the vertex with the third-largest value, or the tenth-largest value, or the $k$-largest value for any $k$.