

A Code Search Engine for Go

Harrison Brewton

Advised by Aws Albarghouthi

April, 2020

“All programming language proofs are by induction”

Creating indexes to aid search through structured data is a finicky business, and it is especially hard to make these indexes achieve sub-polynomial search time. This paper looks into problem of automatically creating an index for arbitrary user defined data types. It details a procedure for the creation of a metric from data types. It then shows how to use this metric to create an index that provides $O(\log(n))$ bounded search. The modularity of this system allowed the creation of module *type2met*, which automatically creates metrics from given types. We then use this module to create a fast search engine for Go functions based on their signature. We demonstrate its use on a number of popular Go modules. We conclude with future work.

1 Background

2 Overview

3 Type2Metric

3.1 Levenshtein Combinator

The Levenshtein edit distance is a commonly used metric for strings. It gives the number of substitutions, deletions, or additions to edit one string into another string [1]. It is well known that the normal Levenshtein distance on strings satisfies the properties of a metric. There are a couple of assumptions that the Levenshtein metric makes: that all substitutions are of the same cost, that deletions and additions are of the same cost and the same cost as a substitution. While this is a safe assumption for strings of characters, it is not general, as there might be better substitutions than others. For example, it is clear that in a real sense $[1, 2, 3]$ is closer to $[2, 2, 3]$ than it is to $[10, 2, 3]$, as a 1 is closer to a 2 than a 10; however, the Levenshtein edit distance would be the same from the first string to the second string (1). To that end we define the Levenshtein Combinator.

Definition 1 (Levenshtein Combinator). *Assume that $d : (T \times T) \rightarrow \mathbb{R}^{\geq 0}$ is a metric. Suppose further that κ is any positive real. Then we say the Levenshtein Combinator of d is $Lev_d : (List\ T \times List\ T) \rightarrow \mathbb{R}^{\geq 0}$. Such that it is defined on lists of T . We define the combinator as:*

$$Lev_d(a, b) = \begin{cases} \kappa|a[1 :]| & |b| = 0 \\ \kappa|b[1 :]| & |a| = 0 \\ \min \begin{cases} Lev_d(a[1 :], b) + \kappa \\ Lev_d(a, b[1 :]) + \kappa \\ Lev_d(a[1 :], b[1 :]) + d(a[0], b[0]) \end{cases} & \text{otherwise.} \end{cases}$$

The Levenshtein Combinator runs almost the same as the traditional Levenshtein edit distance. The only differences are the generalizations of the κ from 1, and the addition of the metric between elements of the list. In fact, the original Levenshtein distance can be recovered by substituting 1 for κ , and the characteristic function $1_{a \neq b}$ as the d metric. We will now show that the Levenshtein combinator of a metric is itself a metric.

Identity of indiscernibles and symmetry are pretty obvious, and so are omitted. The triangle inequality also follows in a similar manner as the proof of traditional levenshtein [2]. Suppose we are considering

a transformation between lists X and Z , that is a series of insertion, deletions, and substitutions. A transformation from X to some Y followed by a transformation from X to some Z is a transformation from X to Z . As the transformation from X to Z is the minimal transformation, it follows $d(X, Z) \leq d(X, Y) + d(Y, Z)$.

3.2 Linear Combinator

We will now briefly show that the linear combination of metrics is a metric. Suppose we have a metric $d(a, b) = \sum c_i d_i(a, b)$, where c_i is a positive real number, and d_i is a metric. Symmetry follows clearly. Identity of indiscernibles follows as each term goes to zero, leaving the sum at zero. The triangle inequality follows $d(a, b) = \sum c_i d_i(a, b) \leq \sum c_i d_i(a, c) + \sum c_i d_i(a, b) = d(a, c) + d(b, c)$.

3.3 Alteration

Suppose that we have a tuple (n, v) , where n is some natural number less than some N , and a mapping $f : n \rightarrow (v \times v \rightarrow \mathbb{R})$. Then we can define the alteration combinator on these tuples.

Definition 2 (Alteration Combinator). *Let $T = E \times V$, such that E is some finite set of objects, and V is any other set of objects. Let $f : E \rightarrow (V \times V \rightarrow \mathbb{R})$ be a relation, such that for all e in E , we have (fe) restricted to $\{(d, v) \in T : d = e\}$ is a metric on that restriction. Furthermore, let $\kappa \in \mathbb{R}^+$. Then we define the alteration combinator $Alt : f \rightarrow T \times T \rightarrow \mathbb{R}^{\geq 0}$*

$$Alt(f, ((e_1, v_1), (e_2, v_2))) = \begin{cases} \min([fe_1](v_1, v_2), \kappa) & e_1 = e_2 \\ \kappa & otherwise. \end{cases}$$

Symmetry, and identity of indiscernibles is pretty straightforward. The triangle inequality for $Alt(f)$, is a little trickier. We consider four cases. Let A, B , and C all be in E . We can consider the triangle inequality as triple of three of these types, and ask where it will hold. If all three are of the same type (A, A, A) , then the inequality is inherited by that of (fA) . Consider (A, A, B) , then the inequality can be seen $(fA)(A, A) \leq \kappa \leq (fA)(A, A) + \kappa \leq (fA)(A, A) + Alt(f)(A, B)$. Consider (A, B, B) , then the inequality can be seen $(fA)(A, B) = \kappa \leq \kappa + (fB)(B, B)$. Consider (A, B, C) , then the inequality obviously holds.

3.4 From Types To Metrics

$$\frac{\frac{x : \text{List } T \quad y : \text{List } T}{\text{Lev}_T(x, y)} \quad \text{LISTS} \quad \frac{x : (T_1, T_2, \dots, T_n) \quad y : (T_1, T_2, \dots, T_n)}{\sum_{i=0}^n c_i d_{T_i}(x_i, y_i)} \quad \text{PRODUCTS}}{\frac{x : T_1 | T_2 | \dots | T_n \quad y : T_1 | T_2 | \dots | T_n}{1_{\text{type}(x)=\text{type}(y)} \min(\kappa, d_{\text{type}(x)}(x, y)) + 1_{\text{type}(x) \neq \text{type}(y)} \kappa} \quad \text{SUMS} \quad \frac{x : \text{Map } T_1 \quad T_2}{\dots} \quad \text{MAP} \quad x : T_1 \quad y : \text{NUMBER}}$$

3.5 Some Additions

3.5.1 Recursive Type Promotion

4 Example: Go Code Look Up

4.1 Generics

5 Metric Trees

Metric trees are an efficient way to search for object based on some metric. We will give a quicker overview of their behavior.

Definition 3 (Metric Tree). *A metric tree is tuple (N, X) , where X is a metric, and N is either nil or a 4-tuple (N, r, v, N) where $r \in \mathbb{R}^{\geq 0}$, and v is some vector, then N 's are defined inductively.*

A metric tree also has two functions related to it. $Index : \text{List } X \rightarrow (N, X)$, which takes a list and converts it to a metric tree. $Query : (N, X) \rightarrow \mathbb{R}^{\geq 0} \rightarrow X \rightarrow 2^X$, which takes a metric tree, a cutoff, and a target, and returns all the nodes in the space closer to the target than the cutoff.

$$Index(xs) = \begin{cases} \text{null} & |xs| = 0 \\ (\text{null}, 0, x, \text{null}) & xs = \{x\} \\ (Index(\{x \in xs' : d(x, p) \leq r\}), r, x, Index(\{x \in xs' : d(x, p) > r\})) & xs = \{p\} \cup xs' \end{cases}$$

$$Query(m, c, t) = \begin{cases} \emptyset & m = \text{null} \\ \bigcup \left\{ \begin{array}{ll} \{v\} & d(v, t) < c \\ Query(a, c, t) & d(v, t) < r + c \\ Query(b, c, t) & d(v, t) \geq r - c \end{array} \right\} & m = (a, r, v, b) \end{cases}$$

6 Implementation

We implemented the automatic type to metric converter, vantage point tree, and Go type extractor in Go lang. We seek to answer the following two research questions:

RQ1 Can we automatically create metrics from given types?

RQ2 Are these metrics effective in reducing time complexity of problems?

7 Related Work

8 Future Work

8.1 Best Approximation

9 Conclusion

References

- [1] Wikipedia contributors. levenshtein distance, 2020. [Online; accessed 19-April-2020].
- [2] Jeremy Kun. Metrics on words, May 2014.