Triadic Memory - Algorithm

Peter Overmann, 08 Aug 2022

This notebook includes the Triadic Memory algorithm in Mathematica language, in the exact form it was originally discovered and published in
https://peterovermann.com/TriadicMemory.pdf .

This version should be useful as a starting point for analyzing the algorithm and for prototyping applications.

For larger-scale applications, there is a Mathematia wrapper for the *triadicmemory* command line tool which runs faster and uses less memory, while providing an identical programming interface.

Triadic Memory Algorithm

```
TriadicMemory[f_Symbol, {n_Integer, p_Integer} ] := Module[ {W, tup},

   (* random sparse vector with dimension n and sparse population p *)
   f[] := SparseArray[ RandomSample[ Range[n], p] → Table[1, p], {n}];

   (* initialize memory *)
   W = Table[ 0, n, n, n];

   (* memory addresses activated by input vectors *)
   tup[x__SparseArray] := Tuples[ Flatten[ #["NonzeroPositions"] ] & /@ {x}];

   (* binarize a vector, using sparsity target p *)
   f[0] = SparseArray[{0}, {n}];
   f[ x_] :=
    Module[ {t = Max[1, RankedMax[x, p]]}, SparseArray[Boole[# ≥ t] & /@ x]];

   (* store {x,y,z} *)
   f[x_SparseArray, y_SparseArray, z_SparseArray] :=
    (++W[[##]] & @@@ tup[x, y, z];);

   (* recall x, y, or z *)
   f[Verbatim[_], y_SparseArray, z_SparseArray] :=
    f[ Plus @@ ( W[[All, #1, #2]] & @@@ tup[y, z])];
   f[x_SparseArray, Verbatim[_], z_SparseArray] :=
    f[ Plus @@ ( W[[#1, All, #2]] & @@@ tup[x, z])];
   f[x_SparseArray, y_SparseArray, Verbatim[_]] :=
    f[ Plus @@ ( W[[#1, #2, All]] & @@@ tup[x, y])];

  ];
```

First steps

Create a memory instance M for vectors of dimension n=500 and a target sparse population of n=5:

```
TriadicMemory[M, {500, 5}];
```

Generate a triple of random sparse binary vectors:

```
{x, y, z} = { M[], M[], M[] }
```

```
{SparseArray[ Specified elements: 10
              Dimensions: {1000} ],
 SparseArray[ Specified elements: 10
              Dimensions: {1000} ], SparseArray[ Specified elements: 10
                                                 Dimensions: {1000} ]}
```

The Hamming distance between two sparse random vectors is about twice their population:

```
HammingDistance[x, y]
```

```
20
```

Store this triple in memory:

```
M[x, y, z]
```

Recall z:

```
M[x, y, _]
```

```
SparseArray[ Specified elements: 10 Dimensions: {1000} ]
```

Hamming distance between z and the recalled value:

```
HammingDistance[M[x, y, _], z]
```

```
0
```

Recall x:

```
HammingDistance[M[_, y, z], x]
```

```
0
```

Recall y :

```
HammingDistance[M[x, _, z], y]
```

```
0
```

The position of vectors within a triple matter:

```
M[y, x, _]
```

```
SparseArray[ Specified elements: 0 Dimensions: {1000} ]
```

This shortcut returns a zero vector:

```
M[0]
```

```
SparseArray[ Specified elements: 0 Dimensions: {1000} ]
```