Connected Components on 1000x1000x1000 Datasets

W. Randolph Franklin (Rensselaer Polytechnic Institute, frankwrATrpi.edu), E. N. Landis (University of Maine, Orono, USA)

Introduction We present an efficient implementation on a laptop computer of the *union-find* algorithm for finding connected components when the input is more than a $1000 \times 1000 \times 1000$ 3D box of binary voxels. Each voxel may be connected either to its 6 orthogonal neighbors, or to all 26 neighbors. Component properties, such as area, and volume are also computed. This implementation is fast enough to permit experimental studies of random connectivity. Determining the connected components of a 2D or 3D array of bits is a required operation in domains such as the following: • Investigating the distribution of cracks in concrete as it is stressed to failure (which is what motivated this research), Nagy et al. (2001), • Extraction of connected components in image processing, and • Determination of porosity of an underground fluid reservoir.

For more details, see (Franklin, 2006).

Prior Art The venerable *union-find* algorithm for finding *connected components* was originally devised to determine sets of equivalenced variables in Fortran. It starts with a set of n ordered elements i, $1 \le a_i \le n$ each in a separate set, $\mathcal{A}_i = \{a_i\}$. Then it repeats a sequence of the following two operations: • Perform a *union* of the sets containing elements a_i and a_j. • *Find* the smallest numbered element in the same set as element a_i. According to Tarjan, the time to execute n *union* and m *find* operations on n elements is $O(n + m\alpha(m, n))$, where $\alpha(m, n)$ is the functional inverse of Ackermann's function. There are many references to the general connected component problem, but fewer to this special case, citations to which may be found by searching on *3D connected components*. However, there do not seem to be implementations on the example size presented here.

Method The key to our implementation is a careful consideration of the data structures, since storing and processing over 10^9 elements has the potential to overwhelm a 32-bit computer. However, our elements, and their equivalence relations, are not general, but are embedded in E^3. Two elements are in the same set *(connected component)* iff they are adjacent, either 6 way (orthogonally) or 26 way (orthogonally or diagonally). Therefore, the most important design decision was not to store elements as individuals, or combined into 2-D regions, but rather as one dimensional *runs*. A run is a set of consecutive voxels with common x and y, and with $z_l \le z \le z_h$. Each run has an i.d. number. As the connected components are determined, runs are formed into trees. A run's father has a larger i.d. than the run.

A run has these components: $(x, y, z_l, z_h, anc/area)$. If this run is the root of its component (as determined so far), then $anc/area$ is the negative of the component's surface area. Else $anc/area$ is the i.d. of this run's father. As the trees are processed, they are flattened so that eventually they have depth 1 and each run's father is the root. The only other nontrivial data structure is a table listing the runs for each (x, y), which is built up as the runs are formed.

As the input data file is read, voxels are grouped into runs. After each run is formed, it is assigned an i.d. in increasing order. After all the runs with a given (x, y) are formed, then some uniting is done. For any run s situated at $(x - 1, y)$ and $(x, y - 1)$ that is adjacent to some run r at (x, y), the root of s is made to point to the root of r. Trees are squashed as they are traversed, so that s and r become no farther than 1 from their common root. If 26-connectivity is being used, then also adjacent runs at $(x - 1, y - 1)$ are linked in. Also, as adjacencies are determined, the component's surface area, stored in the root run, is updated. Because overlapping runs, i.e., adjacencies, affect the area, this is easier to do now than later.

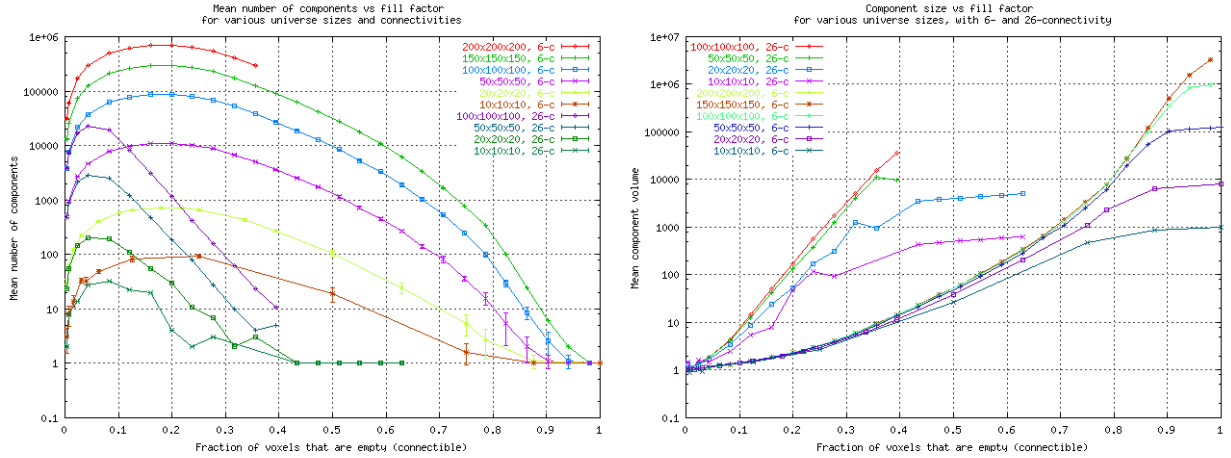
After the input has been read, each component is one tree of runs, but the tree might have a large height. Therefore, one final step passes down thru the array of runs, flattening the trees to height 1. The total execution time is linear in the number of voxels plus runs. Most of the time is spent in reading and processing the input. E^2 is a special case of E^3; processing images with over 18000^2 pixels is easy.

Results The motivating application was Landis's study of how concrete fails under compression, (Landis et al., 2006; Nagy et al., 2003, 2001). 2D cross sections, of which 3 in different directions (from a smaller example) are

shown on the previous page; do not capture the 3D nature of the fractures. Therefore it was proposed that counting connected components might be useful. In this largest example, with a universe of $1024 \times 1088 \times 1088 = 1{,}212{,}153{,}856$ voxels, 50% of the voxels were empty. There were 4,539,562 components, formed from 20,216,828 runs, containing an average of 30 voxels each. During the 29,978,799 times that a pair of adjacent runs were processed to combine their components, the average number of pointers followed per run was only 8.4. Also, in 14,301,533 of those cases, the pair of adjacent runs was already in the same component. The largest component had 2993 runs and a volume of 23675. Many components had only one run and two voxels.

The implementation environment was a 2GHz IBM t43p laptop with SuSE linux and gcc 4.1.0. The virtual memory used, which is a function of the space preallocated for runs and components, which was sized for this example, was only 340MB. The program's elapsed time was equal to its CPU time, which was 51 CPU seconds. Smaller examples run proportionately faster, in perhaps 0.1 seconds for a $100 \times 100 \times 100$ universe. Very complicated cases would run more slowly.

The program is efficient enough to perform repeated experiments to determine the properties of random datasets. So, we asked this question: How many components are generated for various universe sizes and fill factors (probability that a voxel is empty)? We did 10 runs each for many combos of universe sizes and fill factors for 6-connectivity. Then we did 1 run each for some combos of sizes and fill factors for 26-connectivity. The following figure shows the results. The error bars for the 6-connectivity are 1 sigma to each side of the mean. The 26-connectivity cases are the curves scrunched to the left.



We observe that the fill factor giving the maximum number of components is independent of the universe size. For 6-connectivity, p=.2 (approx), gives the most components. For 26-connectivity, p=0.05 does. This independence is reasonable since connectivity is a local property.

We then re-analyzed the above data to show how component size (volume) depended on universe size and fill factor. The lower group of lines is 6-connectivity, while the upper group, which does not extend all the way to the right, is 26-connectivity. The 26-connectivity lines are more irregular since we ran fewer cases.

For component sizes much smaller than the universe size, the universe size was irrelevant, which is reasonable. The limiting case for a fill factor approaching 1 is one component whose size is the universe's volume.

It's not clear what the functional relationship is. In one dimension, component sizes (lengths) would be exponentially randomly distributed, with mean length=$\frac{1}{1-p}$. However, here in 3D, neither the component volumes nor their lengths appear to follow this. This suggests an area for theoretical work.

This research was supported by NSF grants CCR-0306502 and DMS-0327634. (October 27, 2006)

References

Franklin, W. R. (2006), 'CONNECT – find 3D connected components)', `http://wrfranklin.org/pmwiki/ConnectedComponents`.

Landis, E. N., Zhang, T., Nagy, E. N., Nagy, G. and Franklin, W. R. (2006), 'Cracking, damage and fracture in four dimensions', *Materials and Structures* .

Nagy, E., Zhang, T., Franklin, W. R., Nagy, G. and Landis, E. (2003), 3D analysis of tomographic images, *in* '16th ASCE Engineering Mechanics Conference', U Washington, Seattle. electronic proceedings.

Nagy, G., Zhang, T., Franklin, W., Landis, E., Nagy, E. and Keane, D. (2001), Volume and surface area distributions of cracks in concrete, *in* C. Arcelli, L. Cordella and G. S. di Baja, eds, 'Visual Form 2001: 4th International Workshop on Visual Form IWVF4', Vol. 2051/2001 of *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, Capri, Italy.