The spring bounces back: Appendix

Jonathan Bourne

June 2020

This appendix describes the methods used to find the converged positions of the SETSe algorithm substantially faster than using guesswork. The two main methods are auto-convergence, which finds optimal parameters for a given network, and the bi-connected method, which breaks the network into smaller pieces and solves for each piece then re-assembles.

1 Auto-convergence

The SETSe algorithm must be correctly parametrised or the system will not converge. The outcome of the convergence of a SETS embedding is mostly controlled by the relationship between the time step and the coefficient of drag. The four outcomes from SETSe are shown in figure 1. Figure 1 also shows the static force at each iteration of convergence for the Caltech36 Facebook network. The convergence parameters are identical in all cases, apart from the coefficient of drag c. The four outcomes are divergence ($c = 0.65$), failure to converge ($c = 0.7$), noisy convergence ($c = 0.75$) and smooth convergence ($c = 1.2$). Divergence occurs as a result of the discrete time step; if the time step is too large relative to the drag, the system gains energy, causing the kinetic energy and spring energy of the nodes to explode. Failure to converge occurs when the system is in a stasis where it can neither lose energy nor diverge. With noisy convergence, the system loses energy on average, but the system goes through erratic jumps caused by force shocks propagating through the network (discussion of shock propagation is beyond the scope of this paper). As can be seen from the figure, smooth convergence is the most desirable as it reaches the converged state most quickly. It should be noted that in figure 1 the outcomes 'noisy convergence' and 'failure to converge' have two waveforms superimposed on each other. This appears to be due to force shock waves moving back and forth across the network.

Figure 2 shows the final static force versus the coefficient of drag for three time-step values. The axes of the plot are the static force after 3000 iterations and the ratio of the drag coefficient to the time step. The figure shows that the static force is a convex function of the ratio. This convex shape allows a search to be used to find a parametrisation close to the optimum, and is the basis for the auto-SETSe function.

Figure 1: Different outcomes from using SETSe on the same dataset: changes caused by different values of the coefficient of drag.

It should be noted that figure 2 shows that smaller values of Δt give a larger search space to find optimal conditions.

The auto-SETSe algorithm is shown in algorithm 1, which takes advantage of the convex shape of the static force under different drag values. Auto-SETSe uses the same parameters as the SETSe algorithm; in addition, it uses a set of hyper-parameters used to tune the value s. The auto-SETSe algorithm runs in a while loop with several termination conditions, the first of which is that the current iteration p does not exceed the maximum allowed number of iterations $p_{\max}$. This prevents the loop running infinitely.

Simulations are terminated early when the static force exceeds the total absolute force produced by the nodes because, under optimal parametrisation, this does not occur. The simulations are run using a much shorter number of iterations than during normal SETSe. The algorithm searches the log 10 space between user-defined highest and lowest possible drag values. If none of the drag values get static force values lower than the initial value, then the time step is reduced and the process repeated. Once the simulation has found a value of drag and time-step combination that smoothly converges, a binary search is performed to find the optimum value. With the optimum value identified, the simulation is run using a larger number of iterations (of the order of 10 times larger); this final stage adjusts the time-step if the process becomes noisy, which can occur when the static force becomes small compared to its initial value. This approach using early termination for sub-optimal parameters and much shorter simulation times makes finding parameters that are fast and delivering high-

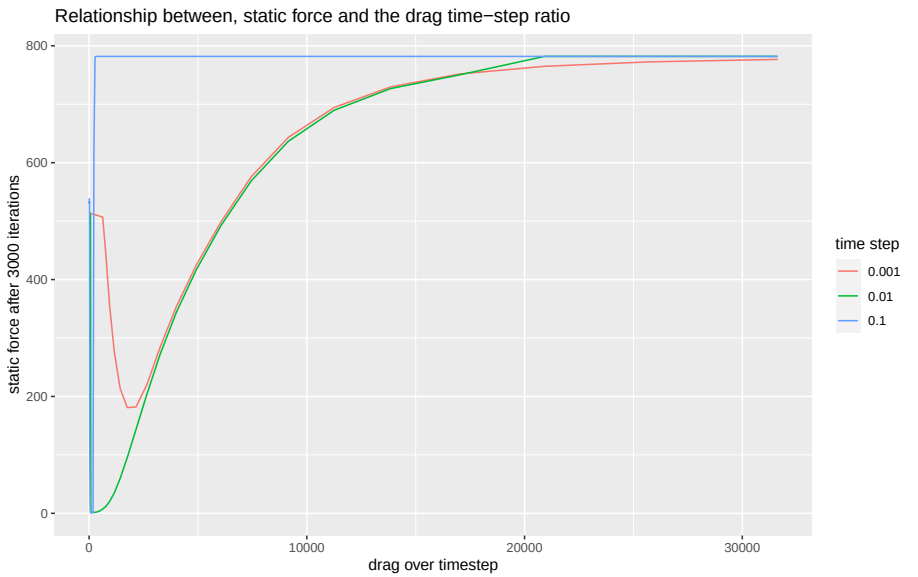

Figure 2: The static force at after a set number of iterations is convex relative to the ratio of drag to time-step size.

quality results straightforward.

The static force after the final iteration of the SETSe algorithm is $\eta = \sum |f_i|$, where η is the sum of the absolute value of the static force experienced by all nodes in the network. The value η_0 is the static force and initialisation. λ is the convergence tolerance; once η has fallen below this level, the system is said to have converged and therefore further parameter search is not required. Auto-SETSe is said to have found optimum parameters when the value of η stabilises. The system is said to be stable when the previous best value of b ($b_{\min}$) and the current value are within ϕ of each other. This is shown on line 9 as b_p, if $\eta_p < \eta_0$ and is twice the tolerance otherwise, shown on line 12. If $p \geq p_{\max}$ or $b_p < \phi$ but the system has not converged, meaning $\eta_p > \lambda$, then SETSe is run again using a substantially larger number of iterations using the best value of c found by the auto-SETSe process.

Whilst this process is not guaranteed to result in a converged SETSe, it provides satisfactory embeddings in most cases.

2 The bi-connected component solution

Although auto-SETSe is very valuable, there remain convergence issues even on ideal parameters. Post-convergence analysis of the Caltech36 network shows the 31 nodes (4% of total graph), which have only a single neighbour, make up 31% of the static force. This is intuitive as nodes on the periphery of the network or

Algorithm 1: The auto-SETSe algorithm.

Result: Graph $\mathcal{G}$ embedded on best value of c for given Δt

$p = 1$;
$s_p = 0$;
while $(\eta_p > \lambda)$ & $(b_p > \phi)$ & $p < p_{max}$ **do**
  $c_p = \frac{\Delta t}{10^{s_p}}$;
  Run SETSe using c_p;
  Store η_p the output of SETSe along with c_p;
  if $\eta_p < \eta_0$ **then**
    Set $s_p + 1$ using the next value produced by a binary search process on the stored values of s;
    $b_p = \frac{\eta_p - \eta_{\min}}{\eta_{\min}}$
  else
    $c_p = 10^{\log_{10}(c_p)+1}$;
    if $c_p >$*max drag* **then**
      $c_p =$ drag min;
      $\Delta t = \Delta t \cdot$ time step shrinker;
    end
    $s_{p+1} = \log_{10}(\frac{\Delta t}{c_p})$;
    $b_p = 2b_{\min}$
  end
  $p = p + 1$;
end
if $\eta_p > \lambda$ **then**
  Run SETSe using best value of c and a larger number of iterations
end

nodes that connect two parts of the network together can be subject to large and unpredictable changes in experienced force, much like a double pendulum. This means that a slight change in position due to group nodes moving can produce substantial static force on these peripheral or joining nodes. The most effective way to solve this is to break the network up into its bi-connected components. A bi-connected component is a sub-graph that will form a separate component with the removal of a single node, called an articulation node. Such nodes act like bridges between groups of nodes and so are prone to experiencing sudden shifts in force magnitude and direction. When a network is decomposed into its bi-connected components, it is called a block-cut tree. An example of a network with four bi-connected components is shown in figure 3. In the figure, articulation nodes are named a_i, where the suffix i indicates the index of the articulation node.

Because the force in SETSe is transferred through the edges, the entirety of the information about the sub-graph is transferred through a single edge. This means that the sub-graph can be simplified to the node that connects it to the rest of the graph (also called the articulation point). The force of the node representing the simplified network is the opposite of the force in the remaining network. By decomposing the network into a block-cut tree made up of all bi-connected components, the degrees of freedom of the network as a whole are greatly reduced. The reduction of the degrees of freedom of the network allows faster convergence of the network as a whole. The bi-connected method functions in a similar way to the Barnes–Hut algorithm [1], which has been applied to reduce the complexity of simulations in astrophysics and has also been applied to some spring embedders [3]. Applying the bi-connected algorithm [2] is efficient as it runs in linear time. The restricted movement and constant force direction allow the bi-connected method to find the final position on the manifold in a way that is not possible for the double pendulum mentioned earlier. The simplification works because only the net force of the other bi-connected components is considered (which is static), and the dynamics are not considered during the solution.

The basic outline of the bi-connected component method is shown in algorithm 2. First, graph $\mathcal{G}$ is decomposed into its bi-connected components. Each component is balanced such that the net force from the connecting component is simplified into the articulation node for that bi-connected component. Then, each bi-connected component is projected into SETSe space. This creates a list of relative node elevations. The function then calls a sub-routine (algorithm 3), which combines all the relative elevations into a single absolute elevation.

A bi-connected component can be connected to many other bi-connected components, and an articulation node can be connected to many bi-connected components. This means the conversion from relative to absolute elevation values must be treated carefully. The algorithm that does this is shown in algorithm 3. This algorithm takes the largest bi-connected component to be the origin point of the network and incrementally adds additional bi-connected components to the origin. This means that any bi-connected component being added will be added directly on to the main network not to another relative

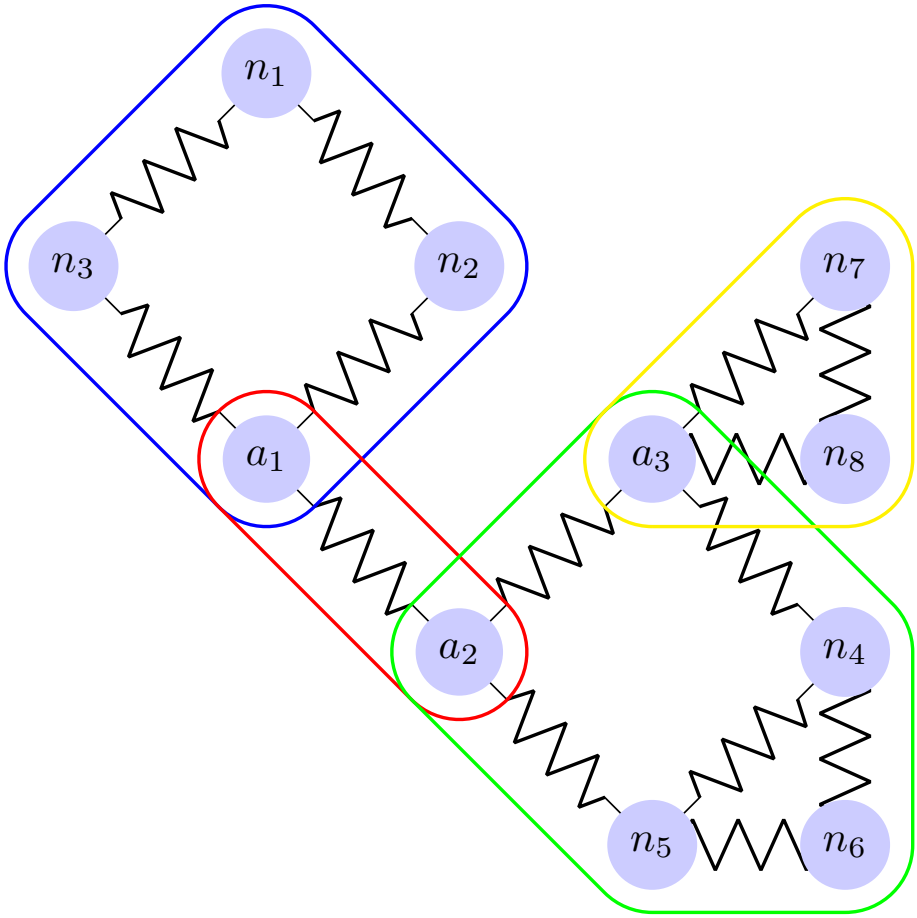

Figure 3: An example network with the bi-connected components highlighted. Articulation nodes are named as a_1, a_2 and a_3; if removed, these nodes cause the network to break into two components. This network includes four bi-connected components.

block. The function takes a list of relative graphs $\mathbf{G}_{\text{vect}}$, then initialises an empty graph $\mathcal{G}_{\text{abs}}$ and an empty vector of articulation points Art_{vect}. It adds the origin block to the empty graph. It then adds all the articulation points to the articulation vector. It then takes all the relative graphs that contain the nth articulation points and converts the relative elevation to absolute by subtracting the elevation of the articulation point on the target bi-connected component G_x from all nodes on G_x and adding the elevation of the same articulation point on G_{abs} to all nodes on G_x. The bi-connected component G_x is then added to a temporary graph $\mathcal{G}_{\text{new}}$ until all bi-connected components connected to the nth articulation point on Art_{vect} have been converted to absolute values. This process continues until all relative graphs have been combined into a single absolute graph.

Algorithm 2: Bi-connected SETSe.

Result: Graph $\mathcal{G}$ embedded on best value of c for given Δt for each bi-connected component of $\mathcal{G}$

find bi-connected components r of $\mathcal{G}$;
Balance the forces between each r such that for each r $0 = \sum f_i$;
Identify largest r and Run auto-SETSe($\mathcal{G}_r$);
$\mathbf{G}_{\text{vect}} = \emptyset$;
for $r \in \mathcal{G}$ **do**
    Run auto-SETSe on all r;
    $\mathbf{G}_{\text{vect}} = \mathbf{G}_{\text{vect}} + \mathcal{G}_{\text{r}}$;
end
$\mathcal{G}_{\text{abs}} = \text{abs\_elevation}(\mathbf{G}_{\text{vect}})$;
return $\mathcal{G}_{\text{abs}}$

The network Caltech36 is in fact over twice as slow as the bi-connected method (although still only 12 seconds). However, nodes with only one neighbour now have close to zero static force (2e^{-10}N), and the overall network has 20% less static force when using the bi-connected method when compared to auto-SETSe. Figure 4 shows the difference in convergence time for the Facebook 100 dataset using auto-SETSE and the bi-connected method. On the y-axis, the figure shows the time taken using the bi-connected over the time taken for the auto-SETSe version. The x-axis shows the difference in hours. It can be seen that the bi-connected version is faster than auto-SETSe 73% of the time. However, when the bi-connected algorithm is slower, the reduction in speed is small in absolute terms. The worst case increases the time by 21 minutes, and the best case decreases the time by 2.3 hours. Overall, using the bi-connected algorithm was a modest 13% faster than using auto-SETSe alone, saving approximately 21.6 hours of computation time. If both algorithms are allowed to run across multiple cores, the bi-connected can be several times faster, from the user perspective. Both approaches successfully converged over 95% of the networks, but needed a small change to the number of iterations used when finding the coefficient of drag for the remaining networks (2000 to 1000 hyper

Algorithm 3: abs_elevation. Combine all bi-connected components into a single structure.

Result: A graph with all embeddings set relative to a single reference value

$\mathcal{G}_{\text{abs}} = \emptyset$;
$\text{Art}_{\text{vect}} = \emptyset$;
$\mathcal{G}_{\text{new}} = \mathcal{G}_1$ #Origin block;
$n = 1$ # The origin block is indexed as 1;
$t = 1$ # vector of indices of blocks to be added
while $n \leq$ *number of blocks* **do**
    $\mathcal{G}_{\text{abs}} = \mathcal{G}_{\text{abs}} + \mathcal{G}_{\text{new}}$;
    $\text{Art}_{\text{vect}} := \text{Art}_{\text{vect}} + \{n_{\text{art}} \in \mathcal{G}_{\text{new}}\}$;
    $\mathbf{G}_{\text{vect}} = \mathbf{G}_{\text{vect}} \setminus t$ #remove targeted blocks from block list ;
    $n = n + 1$;
    #increment the index
    $t = \text{Index}(\{x \subset \mathbf{G}_{\text{vect}} : \text{Art}_{\text{vect}}[n] \in x\})$;
    $\mathcal{G}_{\text{new}} = \emptyset$
    for $x \in t$ **do**
        subtract the elevation of node $\text{Art}_{\text{vect}}[n]$ in $\mathcal{G}_x$ from all nodes in $\mathcal{G}_x$ then add the elevation of $\text{Art}_{\text{vect}}[n]$ in $\mathcal{G}_{\text{abs}}$ to all nodes in $\mathcal{G}_x$.
        $\mathcal{G}_{\text{new}} = \mathcal{G}_{\text{new}} + \mathcal{G}_x$
    end
end
return $\mathcal{G}_{\text{abs}}$

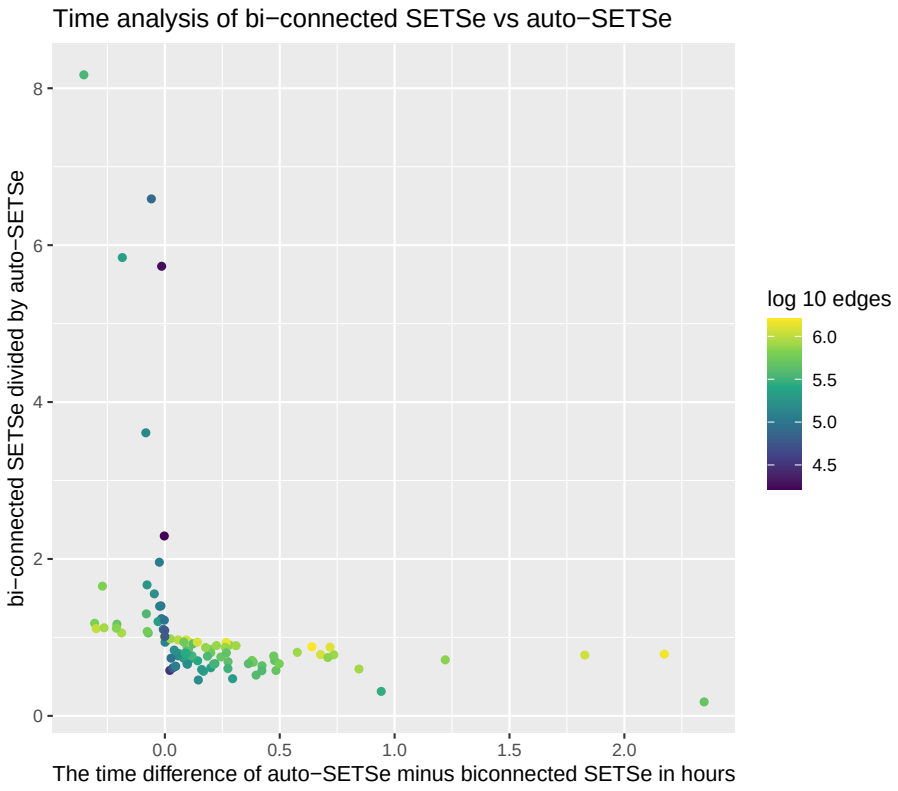

Figure 4: Although the bi-connected version of SETSe is not guaranteed to be faster than auto-SETSe, across multiple networks, the bi-connected method is substantially faster; this is particularly true for larger networks.

iterations). A detailed explanation of when the bi-connected version is faster is beyond the scope of this paper; however, it appears loosely related to the size of the network and the number of bi-connected components.

References

[1] Josh Barnes and Piet Hut. "A hierarchical O(N log N) force-calculation algorithm". en. In: *Nature* 324.6096 (Dec. 1986). Number: 6096 Publisher: Nature Publishing Group, pp. 446–449. ISSN: 1476-4687. DOI: 10.1038/324446a0.

[2] John Hopcroft and Robert Tarjan. "Algorithm 447: efficient algorithms for graph manipulation". en. In: *Communications of the ACM* 16.6 (June 1973), pp. 372–378. ISSN: 00010782. DOI: 10.1145/362248.362272.

[3] Aaron Quigley and Peter Eades. “FADE: Graph Drawing, Clustering, and Visual Abstraction”. en. In: *Graph Drawing*. Ed. by Joe Marks. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 197–210. ISBN: 978-3-540-44541-8.