AMSI
VACATION
RESEARCH
SCHOLARSHIPS
2018-2019

Chemically Feasible Configurations of Topologically Tangled Cubes

Jade Bujeya
Supervised by Dr Toen Castle and
Dr Christopher Lenard
La Trobe University

Vacation Research Scholarships are funded jointly by the Department of Education and Training and the Australian Mathematical Sciences Institute.

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

Australian Government
Department of Education and Training

Abstract

In this report we consider different ways to build a cube from sticks. This is an abstraction of a problem faced by chemists who intend to build a cube-shaped structure from mostly rigid components, which together form the edges of a cube. We consider a variety of isotopically distinct cube structures and analyse the required shape (e.g. straight-line segments joined at an elbow of constant angle) together with the distances between lines which may experience a repulsive force in a chemical situation.

This was achieved by simulating the structures as topologically and geometrically constrained spring networks using a Mathematica program written by the author. Optimal configurations for isotopically distinct cubes were found and properties of these configurations such as symmetry, density, presence of a central void which could contain a solvent molecule etc. were identified allowing predictions about the conditions under which each structure could be formed.

1 Introduction

A naive conception of a cube is as a solid, three-dimensional object. This understanding works well in the realm of Lego City (Lego 2018), but more awesome possibilities exist when we abstract the cube to its vertices and edges, which is to say we use the graph theory definition of the cube. In graph theory, a structure such as a cube is defined completely by its set of vertices, V, and edges, E, where the edges are pairs of points. For our cubes we use the labelling:

$$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$E = \{\{1,2\},\{2,3\},\{3,4\},\{4,1\},\{1,5\},\{2,6\},\{3,7\},\{4,8\},\{5,6\},\{6,7\},\{7,8\},\{8,5\}\}.$$

This definition of a cube is much broader than the traditional concept of the solid. However graph theory tells us nothing about the actual arrangement of the edges in space.

Figure 1 shows three different arrangements of points connected by lines corresponding to the vertices and edges of the definition above. The differences are described using the concept of 'isotopy'. The configurations (a) and (b) from Figure 1 are isotopic, as they can be continuously deformed into one another without lines passing through each other. In contrast (b) and (c) are not isotopic, as (c) cannot be continuously deformed into either (a) or (b) due to the presence of a Hopf link (see below) in this embedding. For this reason we say configurations (a) and (b) are the same isotope and (c) is a distinct isotope.

The term 'Hopf link' refers to the simplest arrangement of two interlinked cycles. It is simply one example of the infinite number of possible forms of tangling, examples of which are shown in

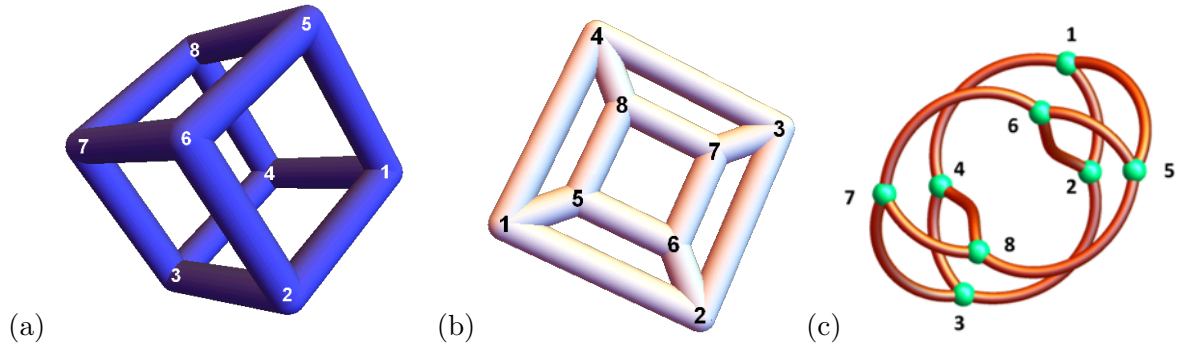

Figure 1: Three arrangements of a cube in space. (a) The traditional block cube without its faces and body. This is called a 'wireframe'. (b) A deformation of (a) into the plane shows that the cube is a planar graph. (c) A cube – with identical connectivity to (a) and (b) – can be constructed in a way that cannot be deformed into a traditional layout without lines passing through each other.

Figure 2.

From a topological standpoint, a link refers to any number of cycles which cannot be continuously deformed apart without lines passing through each other, while a knot is a single component link which cannot be continuously deformed into a circle without having to pass through itself. Knots and links are important as the differences between different knots and links allows us to identify unique cube isotopes. This can be done by examining each cycle in the graph, and each combination of disjoint cycles, to find any knots or links present within the cube embedding. A different collection of knots and links indicates that two cube embeddings are distinct isotopes.

Beware that "isotope" is a word with a different meaning in chemistry. This report deals with chemistry insofar as it researches chemically reasonable structures, however we use the word in the topological sense just defined. Topological isotopes have chemical significance, as although they each contain the same basic components, different constructions of the same materials can have vastly different chemical properties (Brasher, Scharein and Vazquez 2013).

The objective of this project is to explore low energy configurations of tangled isotopes. The energy function implicitly described in the method rewards configurations which have edges of a constant predetermined length and shape, while also avoiding edges being crammed together. These features were chosen as they are consistent with simple chemical self-assembly from long chain molecules. There are an infinite variety of tangled cubes, and indeed of any cyclic graph due to the possibility of arbitrary knotting in any cycle, so it is necessary to restrict the possible tangling of the cubes as was done by Castle (2013) in the PhD thesis "Entangled graphs on surfaces in space". This restricts the

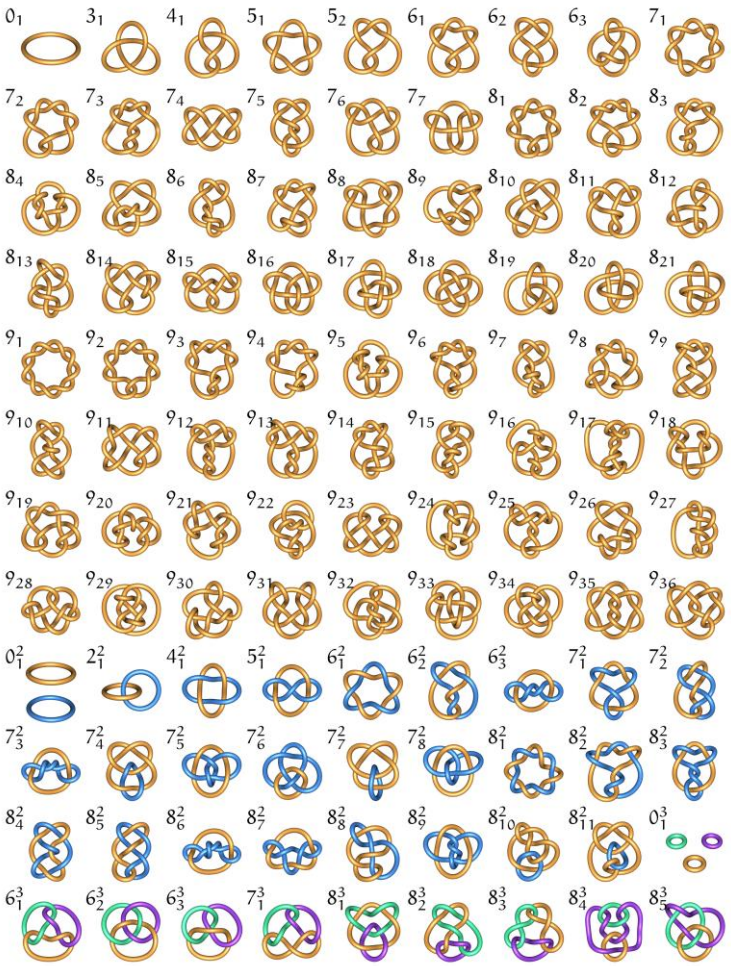

Figure 2: A table of knots and links of up to three cycles and nine crossings (Scharein 2018).

class of configurations to those which can reticulate a torus (a natural way of limiting complexity) and was further restricted to isotopes containing simpler torus knots and torus links. The specific isotopes explored are listed in Figure 1, from Castle (2013), Table 2.13.

For this project, optimal configurations of each isotope were found by creating and using a Mathematica computer program which modelled the molecular configurations as a network of connected springs. It was designed to adjust the length of line segments in the embedding, the distance between line segments, and the angle between some of these segments. The aim was to find a specific arrangement of points and lines in space that respected the geometrical constraints that a chemical version of these cubes would experience.

Energy	Name	Tiling	Side Vectors	Knots	Links
0.83	F	<4,4,4,12>	(1,0),(0,1)	-	-
1.45	G	Sphere	-	-	-
1.33	A	Honeycomb	(1,0),(0,1)	-	(2,2)
1.76	C	Brick Wall	(1,0),(1,1)	(3,2)	2*(2,2)
2.20	B	Brick Wall	(1,2),(0,1)	-	(2,4)
2.31	D	Honeycomb	(1,1),(1,0)	2*(2,3)	(2,2),(2,4)
2.82	H	<4,4,6,10>	(1,1),(1,2)	4*(3,2)	(2,2)
2.91	I	Brick Wall	(1,0),(2,1)	(3,2),(5,2)	2*(4,2)
2.98	J	<4,4,6,10>	(1,1),(2,1)	4*(3,2)	(4,2)
3.14	K	<4,4,6,10>	(3,1),(1,0)	-	(6,2)
3.53	E	Brick Wall	(1,2),(1,1)	4*(2,3),(3,4)	

Figure 3: A table modified from Castle (2013) which includes the names, knots, and links of each of the cube isotopes.

2 Method

Initial configurations of each isotope were provided by my supervisor, Dr Castle. These were consistent with the isotopes published in his PhD thesis 'Entangled graphs on surfaces in space'. We created a Mathematica program to process this data and to rearrange the vertex and edge locations so as to continuously deform the configuration. This was done in such a way as to create more space between line segments in order to mimic chemical repulsions, while also achieving uniform line length.

The process can be understood as a quasistatic 'spring model', where physical forces are calculated and the system evolves in a number of snapshots, without momentum. More specifically, we calculated a series of forces on each point due to line segment length, distance between segments, and angles between segments of edges. Each of these forces were determined by separate procedures within the program, before being compiled together by a final function. The final function then moved the points according to the calculated forces. This process was repeated through a pre-set number of iterations, with the locations of the points recorded at each iteration to produce sequential lists of locations.

2.1 Repulsion

The first stage of the repulsion procedure was to measure the shortest distance between line segments. Initially the closest point on each infinitely extended line was calculated. As our segments were of finite length it was necessary to then use this information to deduce the shortest distance between segments, which in some cases occurred between segment ends.

This problem was originally addressed using simple Mathematica code, however as the project progressed, it became obvious that for such a fundamental and frequently used function, faster code would be needed. For this reason, a section of C-code published by username tintin (2014) was used, however when this too was found inadequate, a compiled version of the same code by username Jason B. (2017) was implemented.

This function was then used to find the distances between each line segment and every other line segment in the embedding. The difference between this distance and the pre-set minimum repulsion distance was then used as a force on the line segments along the vector passing through them, which dropped off once the segments were more than the required minimum distance apart. At this stage, it was necessary to specify that only non-adjacent line segments were to be included in the calculation, in order to avoid lines being repelled by those with which they shared a point.

2.1.1 Maintaining the Topology

While these values were being calculated, a recording of the minimum distance between a line segment and any other line segment was made, and used to restrict the motion of that line segment. Early on, this restriction was simply based on the minimum distance between any two line segments in the embedding, but it was later decided to use unique vales for each line rather than a single value, and restrict the motion of each line individually rather than the embedding as a whole, in order to allow more movement in each iteration of the program while still preventing line segments from crossing one another.

2.1.2 Correction and Transfer of Force Vectors

After this, we included a correction function, which would approximate the force on a line to be zero if the actual force was less than 10^{-6}, in order to prevent the program from wasting time calculating ridiculously small values likely caused by and possibly propagating rounding errors

2.2 Length Correction

The second procedure controlled segment length correction. Each segment grew or contracted towards its pre-set length by movement of each endpoint along the vector passing through the segment. This motion was then scaled according to the discrepancy between the segment length, and the desired length. The forces on each point due to length correction of its adjacent segments were then summed and scaled down, to avoid these movements dominating the repulsion and angle correction forces on

the embedding.

2.3 Angle Correction

The third procedure created and maintained the specified angle in the elbows positioned at the centre of each edge of the embedding. The procedure first calculated the angle between the segments, and then applied a scaled force on the two points not common to the segments along the respective tangents to the arc between these points. As with the above, this action was repeated in each iteration until the desired angle was reached.

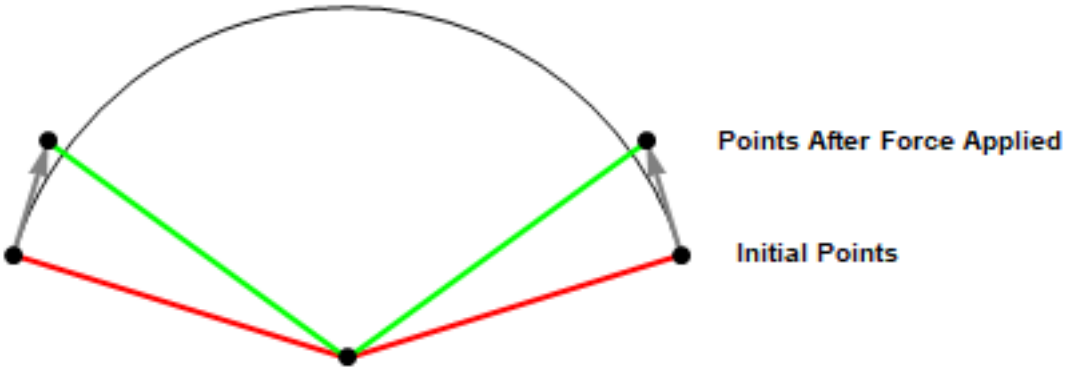

Figure 4: A visual explanation of the Angle Correction procedure.

2.4 Repeat and Run

The repetition function ran the above procedures, tabulating the point locations at each iteration. This allowed for the creation of an animation showing the evolution of the embedding over time, as well as graphs displaying the various forces on the final embedding due to each of the force procedures.

3 Results

Each of the non-trivial embeddings were tested using this program in order to find embeddings conforming to the pre-set constraints on segment length, repulsion, and elbow angle. Embeddings F and G were ignored, as they contain no tangling. The others were allowed to run with an initial repulsion distance of half the line length (corresponding to a volume of exclusion around each line in the shape of a cylinder with radius one quarter of the line length) until they appeared to have settled, meaning there had been no significant changes to the embedding in at least 200 iterations. Each was then

run for a further hundred iterations at progressively smaller repulsion distances in order to identify the threshold repulsion distance below which further decreases of the excluded cylinder radius had no effect. This behaviour is summarised in Figure 5.

	A	B	C	D	H	I	J	K	E
50%	0.00043	0.0115243	0.00985746	0.00616602	0.0863642	0.118573	0.103475	0.119837	0.137749
40%	2.81E-05	0.00548251	0.0046545	0.00307711	0.0520522	0.0692308	0.0515748	0.0793063	0.07841154
30%	2.81E-05	0.00216481	0.001953934	0.00156006	0.00702087	0.037191	0.00655819	0.03618478	0.01516519
27%	2.81E-05	0.00106046	0.000975947	0.0015088	0.00409104	0.03099597	0.00594795	0.02508504	0.01193284
25%	2.81E-05	0.000591192	0.000588307	0.0015088	0.00269412	0.03074681	0.0055557	0.02799103	0.01427971
23.50%	2.81E-05	0.000472801	0.000588307	0.0015088	0.00174859	0.02768976	0.00506006	0.01883293	0.01007069
22%	2.81E-05	0.000472801	0.000588307	0.0015088	0.00154243	0.02579887	0.00402993	0.0191268	0.00943233
20%	2.81E-05	0.000472801	0.000588307	0.0015088	0.00117211	0.01880588	0.00350789	0.01161312	0.01227039
10%	2.81E-05	0.000472801	0.000588307	0.0015088	0.000857377	0.00713305	0.00283635	0.0077818	0.00624457
5%	2.81E-05	0.000472801	0.000588307	0.0015088	0.000857377	0.00728426	0.00252301	0.00720233	0.0046004
1%	2.81E-05	0.000472801	0.000588307	0.0015088	0.000857377	0.00178003	0.0020709	0.00758182	0.00328625
0.01%	2.81E-05	0.000472801	0.000588307	0.0015088	0.000857377	0.00089048	0.00215603	0.00716993	0.00299395

Figure 5: A table showing the overall force on the embedding 100 iterations past its settling point from all of the procedures conbined. Yellow indicates the point beyond which the overall force is no longer decreasing. Red indicates that the overall force is greater than 0.01. Percentages refer to the ratio of line length to repulsion distance. Letters A, B, C, etc. refer to the different isetopes.

4 Discussion and Conclusion

In this report we have shown the most chemically feasible configurations of tangled cubes containing simple forms of tangling. As the repulsion function was the primary area of exploration, and in itself was monotonic, it is of no surprise that the results showed a mostly monotonic relationship between it and the energy function.

It should be noted that the selection of a linear relationship may not necessarily have been the most accurate way to simulate the forces on embeddings. Given the required distance of separation, and the distance between line segments, essentially any function could have been used to approximate the force function. A linear function was chosen in this case for its simplicity and because it offers the most accurate approximation that would not create computational bottlenecks.

The variation from monotonicity seen in the results indicates that the other (non-repulsive) forces on the embedding had a noticeable effect. These variations are likely due to the reduced role that a smaller repulsion distance plays in the deformation of the embedding, so that isotopes got stuck in local minima without the greater force of repulsion to dislodge them.

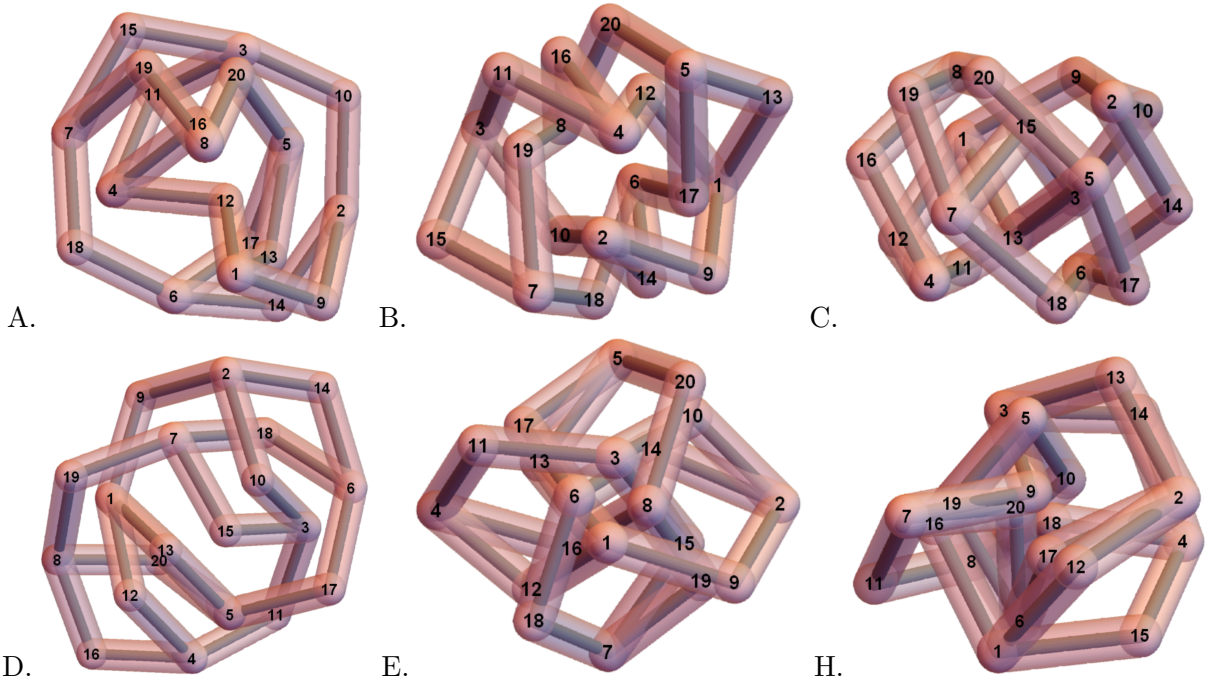

Figure 6: Three-dimensional images of isotopes A, B, C, D, E, and H.

For the purposes of this project it was necessary to ignore repulsion between adjacent edges in order to simplify the programming aspect. In a chemical system this would not be the case. This problem was ameliorated around elbows by the specification of angles between line segments. However this would not have had any effect on the problem for line segments meeting at vertices.

In a chemical system it is possible that structures could occur with a higher genus of tangling than the toroidal configurations considered here. However these configurations are typically more complex and less symmetric, and thus involve more structurally diverse roles: consider some knots in Figure 2 with both highly coiled sections and long joining strands. A component is unlikely to have low energy states in both tightly coiled (helical) and comparatively straight configurations. Thus these embeddings are energetically unflavoured for chemical self-assembly.

The final configurations, as shown in Figure 6, allow some predictions about feasibility for chemical construction. We can deduce firstly that isotope A could be constructed of comparatively thicker, or shorter, molecules or ligands, with D being a close second in this regard. Others require thinner ligands, however their construction might be favoured for other reasons. The space inside B suggests that it might be uniquely suited to formation around a larger solvent molecule, while configuration C might have a higher yield under greater pressure, as it makes a more economic use of space than do A or D. It is also worth noting that Figue 5 shows final energy states and ignores possible higher energy

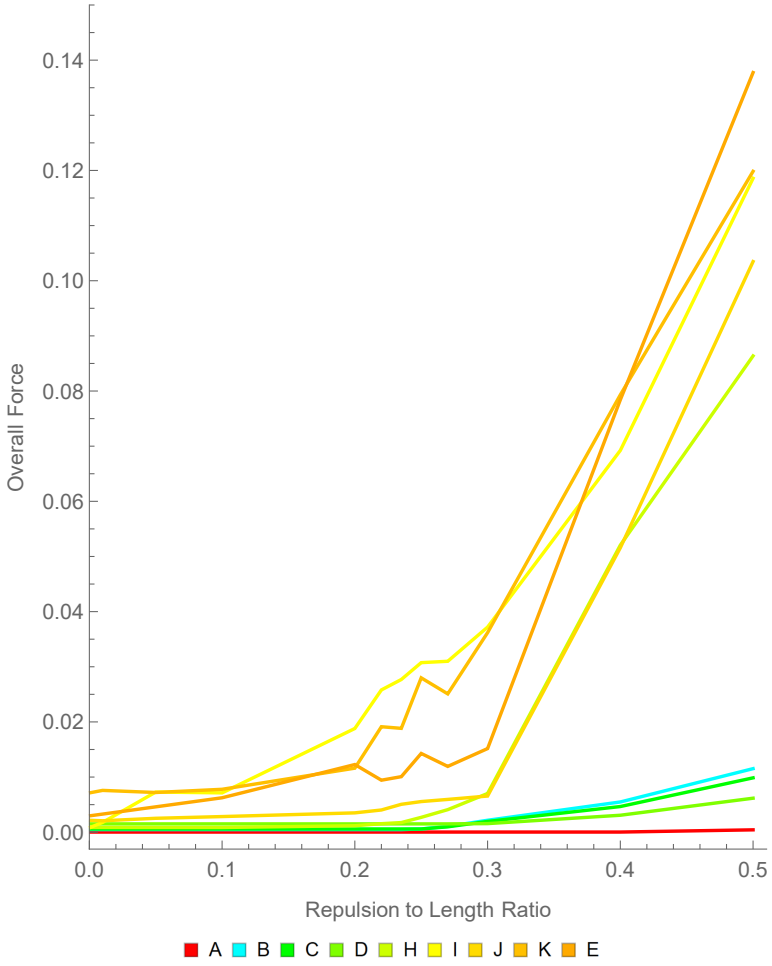

Figure 7: A graph showing the overall force on each line compared to the line length to repulsion ratio.

pathway states involved in the self-assembly process. Of course, it is immediately evident that further work is needed to explore low energy configurations of isotopes I, J, and K. This method could also be used with equal validity for other two or three dimensional embeddings, however these embeddings fall outside the scope of this project, and so were not explored here.

In any case, the author would advise whomsoever might wish to chemically produce any of the structures discussed in this report to consider more research with stricter parameters. This research has used generic structures, such as using chemically typical elbow angles, and so more accurate structural predictions could be achieved with more specific details. We eagerly await your results.

5 Acknowledgements

This research builds on the work of Dr Castle in his PhD thesis 'Entangled graphs of surfaces in space'.

This project would not have been possible without the support and encouragement of the Australian Mathematical Sciences Institute (AMSI) and the Department of Mathematics and Statistics and La Trobe University, Bendigo.

Special thanks to my supervisor Dr Toen Castle, for his unending patients and support as I struggled to learn programming and understand various aspects of academic research during this project, as well as for supplying the data with which I worked and several useful pieces of code.

References

[1] Castle, T 2014, "Entangled graphs on surfaces in space", PhD thesis, Australian National University, Canberra, viewed 11 February 2019, https://openresearch-repository.anu.edu.au/handle/1885/11978.

[2] Lego 2018, *Lego City*, Lego, viewed 22 February 2019, <https://www.lego.com/en-us/themes/city>.

[3] Brasher, R, Scharein, RG & Vazquez, M 2013, 'New biologically motivated knot table', *Biochemical Society Transactions*, vol. 41, no. 2, pp. 606-611.

[4] Scharein RG 2018, *A Knot Zoo*, image, The Knot Plot Site, viewed 22 February 2019, <https://knotplot.com/zoo/>.

[5] Tintin 2014, *Distance between two line segments in 3-space*, Stackexchange, 3 April, viewed 22 February 2019, <https://mathematica.stackexchange.com/questions/45265/distance-between-two-line-segments-in-3-space>.

[6] Jason B. 2017, *Distance between two line segments in 3-space*, Stackexchange, 28 January, viewed 22 February, <https://mathematica.stackexchange.com/questions/45265/distance-between-two-line-segments-in-3-space>.

6 Appendix

```
Jade Bujeya, February 2019.
distSegToSegComp from tintin/Jason B. on Stackexchange.
<https://mathematica.stackexchange.com/questions/45265/
distance-between-two-line-segments-in-3-space>

Relaxer Program:
House Keeping



distSegToSegComp =
  Compile[{{l1, _Real, 2}, {l2, _Real, 2}},
   Module[{small = 10^(-14), u, v, w, a, b, c, d, e, D, sc, sN, sD, tc, tN,
     tD}, u = l1[[2]] - l1[[1]];
    v = l2[[2]] - l2[[1]];
    w = l1[[1]] - l2[[1]];
    a = u.u;
    b = u.v;
    c = v.v;
    d = u.w;
    e = v.w;
    D = a*c - b*b;
    sD = sc = sN = D;
    tD = tc = tN = D;
    If[D < small, sN = 0;
     sD = 1.;
     tN = e;
     tD = c, sN = b*e - c*d;
     tN = (a*e - b*d);
     If[sN < 0., sN = 0.0;
      tN = e;
      tD = c, If[sN > sD, sN = sD;
```

```
        tN = e + b;
        tD = c;]]];
    If[tN < 0, tN = 0;
     If[-d < 0, sN = 0, If[-d > a, sN = sD, sN = -d;
       sD = a]], If[tN > tD, tN = tD;
      If[-d + b < 0, sN = 0, If[-d + b > a, sN = sD, sN = -d + b;
        sD = a]]]];
    sc = If[Norm[sN] < small, 0, sN/sD];
    tc = If[Norm[tN] < small, 0, tN/tD];
    N[w + sc*u - tc*v]]];

linsByPnts[pnts_, lins_] :=
  pnts[[#]] & /@ lins (* gives the endpoints of each line *);

reArrangeResTable[pnts_, lins_, resTable_] := Module[{newList, finResTable},
    newList = {};
    Table[If[lins[[j, 1]] || lins[[j, 2]] == i,
      AppendTo[newList, resTable[[j]]]], {i, Length[pnts]}, {j,
      Length[resTable]}];
    finResTable = Min /@ newList;
    finResTable
    ]; (* this function table in the minimum distances between lines and \
translates them into the minimum distances point can move, to be used later \
in deltas *);

fixUp[stupidlyFormatted_] :=
  Table[ToExpression /@ (Table[
      StringJoin[stupidlyFormatted[[j, i]], stupidlyFormatted[[j, i + 1]],
       stupidlyFormatted[[j, i + 2]]], {i, 1, Length[stupidlyFormatted[[j]]],
       3}]), {j, 1, Length[stupidlyFormatted]}];
```

Swinging (angles)

```
swing1El[pnts_, lins_, {{a_, b_}, {c_, d_}}, optAngle_, optLength_] :=
  Module[{vecbylins, lin1, lin2, normvec, v1, v2, angleDiff, vc, base},
   vecbylins =
    Flatten[Cases[{{{a, b}, {c, d}}, {{b, a}, {c, d}}, {{a, b}, {d, c}}, {{b,
         a}, {d, c}}}, {{_, g_}, {_, g_}}], 1]; (*
   this make sure each vector points to the agnle relivent vertex *)
   lin1 = pnts[[vecbylins[[1, 2]]]] - pnts[[vecbylins[[1, 1]]]];
   lin2 = pnts[[vecbylins[[2, 2]]]] - pnts[[vecbylins[[2, 1]]]];
   normvec = Cross[lin1, lin2];
   angleDiff = VectorAngle[lin2, lin1] - optAngle // N;
   (* a positive angleDiff indicated that we want it to shrink,
   negiative to grow *)
   v1 = -0.2*angleDiff*Norm [lin1]/optLength*Normalize@Cross[normvec, lin1];
   v2 = 0.2*angleDiff*Norm[lin2]/optLength*Normalize@Cross[normvec, lin2];
   vc = -(v1 + v2);
   (* v1 and v2 are the force vector acting on the points *)
   base = With[{i = Dimensions[pnts]}, Table[0, i[[1]], i[[2]]]];
   base[[vecbylins[[1, 1]]]] = v1;
   base[[vecbylins[[2, 1]]]] = v2;
   base[[vecbylins[[1, 2]]]] = vc;
   If[vecbylins[[1, 2]] != vecbylins[[2, 2]],
    Print["Error: see swing part 1"]; Abort[]];
   base
   ]; (* give forces on all points due to one elbow *)

calSwing[pnts_, lins_, ngls_, optAngle_, optLength_] := Module[{},
   Total@Table[swing1El[pnts, lins, lins[[i]], optAngle, optLength], {i, ngls}]
   ]; (* gives all forces due to elbows *)

Pushes
```

```
calPushes[pnts_, lins_, mindist_] :=
  Module[{accepted},
   accepted =
    Complement[Range[Length[lins]], #] & /@ (*
     the 'accepted' lines are those which do not share endpoints with the \
line in question.
     It's a table. *)(Union /@
       Table[Position[lins, lins[[n, 1]]][[;; , 1]]~Join~
         Position[lins, lins[[n, 2]]][[;; , 1]], {n, 1, Length[lins]}]);
   Table[Total[Table[
      Module[{minVec =
         distSegToSegComp[linsByPnts[pnts, lins][[n]],
          linsByPnts[pnts, lins][[i]]]},
       If[Norm[minVec] < resTable[[n]], resTable[[n]] = Norm[minVec]]
       (* If[resTable[[n]]\[LessEqual] 10^-10 , Print[
       "Error: lines too close"]];*);
       If[Norm[minVec] <= mindist,(*
        scaling inverstion now correct DO NOT TOUCH THIS *)(mindist -
           Norm[minVec]) Normalize[minVec],
        Table[0, Length[pnts[[1]]]]] , {i, accepted[[n]]}]], {n, 1,
     Length[linsByPnts[pnts,
       lins]]}]];(* is a compacted version of the original seperation of \
lines fuction. go look at that if you're confused. Otherwise, its giving you \
a table of all the forces on each line due to all the other line accept thos \
with whom it shares endpoints *)

scalePushOnLines[pnts_, lins_, mindist_] :=
  Module[{tempResults = calPushes[pnts, lins, mindist]},
   Table[If[Norm[tempResults[[j]]] <= 0.00001,
     Table[0, {i, 1, Length[pnts[[1]]]}], tempResults[[j]]], {j,
     Length@tempResults}(*this gives linear force, could improve later.
    No actual scaling happening here anymore, all in calPushes.
```

```
    It's a table. *)]];

pushLineToPushPnts[pnts_, lins_, mindist_, optLength_] :=
  Module[{forceOnPnts, forcesOnLines = scalePushOnLines[pnts, lins, mindist]},
   forceOnPnts = Table[0, {j, 1, Length[pnts]}, {i, 1, Length[pnts[[1]]]}];
   Do[forceOnPnts[[lins[[i, 1]]]] =
     forceOnPnts[[lins[[i, 1]]]] + forcesOnLines[[i]];
    forceOnPnts[[lins[[i, 2]]]] =
     forceOnPnts[[lins[[i, 2]]]] + forcesOnLines[[i]], {i, 1, Length[lins]}];
   forceOnPnts];

Pulls

calPullMags[pnts_, lins_, opLength_] :=
  Table[opLength -
    Norm[{linsByPnts[pnts, lins][[i, 2]] -
       linsByPnts[pnts, lins][[i, 1]]}], {i, Length[lins]}];

linsVecs[pnts_, lins_] :=
  Table[linsByPnts[pnts, lins][[i, 2]] - linsByPnts[pnts, lins][[i, 1]], {i,
    1, Length[linsByPnts[pnts, lins]]}] (* gives each line as a vector *);

locs[pnts_, lins_] :=
  Table[Position[lins, i], {i, 1, Length[pnts]}]; (* {line, position} *)

pullVecOnPnts[pnts_, lins_, opLength_] := Table[Which[
       #[[2]] == 1,
       -calPullMags[pnts, lins, opLength][[#[[1]]]]*
        linsVecs[pnts, lins][[#[[1]]]],
       #[[2]] == 2,
       calPullMags[pnts, lins, opLength][[#[[1]]]]*
        linsVecs[pnts, lins][[#[[1]]]], True,
```

```
       Print["Error: see pullVecOnPnts"]; Abort[];] & /@ (locs[pnts, lins][[
       j]]), {j, 1, Length[pnts]}] //
   N; (* collects all the vector forces from the pull that act on each point, \
signs are currently correct *)

combinePulls[pnts_, lins_, optLength_] :=
  0.3*Total[#] & /@ pullVecOnPnts[pnts, lins, optLength] //
   N(* sums element of pull to one vector *);



Combine and Run

deltas[deltaList_, pnts_, lins_, resTable_] := Module[{lengths, res},
   lengths = Norm /@ deltaList;
   res = reArrangeResTable[pnts, lins, resTable];(*res does points*)
   (*1.*)1*
    Table[If[lengths[[n]] > res[[n]]/2.1,
      Normalize@deltaList[[n]]*(res[[n]]/(2.1)), deltaList[[n]]], {n,
      Length[deltaList]}]] (* this function takes in all the forces on the \
lines/pnts and scales them down so that the larhes value in the set deltalist \
is equal to half the smelst distance between any two line sin the set. 1. \
artificial value to reduce movement in each step *);

combineAndRep[pnts_, lins_, ngls_, syms_, mindist_, optLength_, optAngle_,
   reps_] := Module[{workingPnts, data},
   data = {pnts};
   workingPnts = pnts;
   resTable = Table[Infinity, Length[lins]];
   Do[
    workingPnts +=
     deltas[pushLineToPushPnts[workingPnts, lins, mindist, optLength] +
       combinePulls[workingPnts, lins, optLength]  +
```

```
        calSwing[workingPnts, lins, ngls, optAngle, optLength] +
        symFunc[pnts, syms], pnts, lins, resTable];
     AppendTo[data, workingPnts], reps];
    output = data;] ;

makeImageRelaxer[lins_, ngls_, syms_, mindist_, optLength_, optAngle_,
  imageScaler_,
  output_] := {Animate[
   Show[{Graphics3D[{With[{t = Norm[#[[1]] - #[[2]]] - optLength},
          If[t > 0, Blend[{RGBColor[1, .992, .816], Blue}, 2 t],
           Blend[{RGBColor[1, .992, .816], Red}, -2 t]]], Tube[#, 0.1]}] & /@
      linsByPnts[output[[j]], lins],
     Graphics3D[
      Table[Text[Style[i, Bold, Green], output[[j, i]]], {i,
        Length[output[[j]]]}]]}, Axes -> True, AxesLabel -> {"x", "y", "z"},
    PlotRange -> imageScaler {{-1, 1}, {-1, 1}, {-1, 1}}], {j, 1,
    Length@output, 1}],
  sadPush = pushLineToPushPnts[output[[-1]], lins, mindist, optLength];
  sadPull = combinePulls[output[[-1]], lins, optLength];
  sadSwing = calSwing[output[[-1]], lins, ngls, optAngle, optLength];
  sadSym = symFunc[output[[-1]], syms];
  TextGrid[{{"Mean Repulsion Divergence",
     Mean[Norm /@ sadPush]}, {"Mean Length Divergence",
     Mean[Norm /@ sadPull]}, {"Mean Angle Divergence",
     Mean[Norm /@ sadSwing]}, {"Mean Symmetry Divergence",
     Mean[Norm /@ sadSym]}, {"Overall Happiness",
     Mean[{Mean[Norm /@ sadSwing], Mean[Norm /@ sadPush],
       Mean[Norm /@ sadPull](*Mean[Norm/@sadSym]*)}]}}],
  Histogram[Labeled[Norm /@ sadPush, "Repulsion Divergence", Above],
   Length[output[[1]]], ChartStyle -> Red, ImageSize -> Medium],
  Histogram[Labeled[Norm /@ sadPull, "Length Divergence" , Above],
   Length[output[[1]]], ChartStyle -> Green, ImageSize -> Medium],
```

```
Histogram[Labeled[Norm /@ sadSwing, "Angle Divergence" , Above],
 Length[output[[1]]], ChartStyle -> Blue, ImageSize -> Medium],
Histogram[Labeled[Norm /@ sadSym, "Symmetry Divergence" , Above],
 Length[output[[1]]], ChartStyle -> Purple, ImageSize -> Medium]}
```