

# Create molecules with a particular boiling point

Maiyun Zhang

---

*This project implements a method to identify alkanes with a boiling point near a given value based on statistical and topological methods. To achieve this, we use the Wiener index to predict alkane's boiling points structurally and identify alkanes by either brute-forcing or searching with an optimized searcher.*

---

## Introduction

A new chemical's physical properties are usually useful to obtain. A chemical's physical properties depend not only on its composition but also on its structure and the types of bonds. While it is possible to model liquids and compute theoretical boiling points with physical chemistry or thermodynamics, one can also use topological chemistry to model chemicals' behaviour with mathematical approaches and, in turn, predict their physical properties.

Quantitative structural-property relationship (QSPR) is the study to quantify chemical's properties with mathematical models. In 1947, Harry Wiener defined the Wiener index for molecules, thus creating a new way to look at chemicals through the topological properties of the molecule - one of the popular QSPR methods. He demonstrated a correlation between the boiling point of paraffin, a group of organic compounds, and the corresponding Wiener index of the molecular graph, enabling the prediction of the various physical properties based on the molecular structure.

However, the reverse problem, finding chemicals based on known physical properties, is also often helpful for someone trying to identify chemicals or looking for new chemicals.

In this project, I intend to explore ways to identify ways to find chemicals based on some given physical properties with the help of the topological modelling of molecules. Due to the inherent complexity of different types of chemicals, I will limit the scope of this project to alkanes, the same group of chemicals studied by Wiener. The boiling point of molecular chemicals depends on their masses and the strength of the intermolecular forces. Since alkanes are non-polar chemicals, the London dispersion force is the dominant intermolecular force. Its strength mainly depends on the polarizability of the molecule: the less compact it is, the more likely a temporary dipole can form on it. Since the Wiener index effectively quantifies the compactness, it is an adequate property for predicting the boiling point of non-polar chemicals.

# Predicting Boiling Point

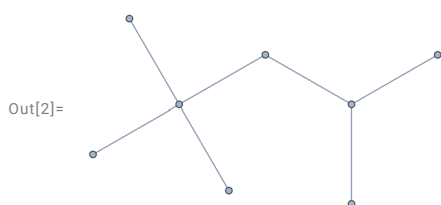
## The Wiener Index

The **Wiener index** (originally called the “path number” by Wiener) is defined as the sum of the minimum distances between all pairs of carbon atoms. Although originally defined for studying chemicals, it is now known as a generic topological index.

It is also more convenient to represent molecules as graphs. This way, all carbon atoms are simply shown as the vertices of the graph.

Example:

```
In[1]:= exChemical = Entity["Chemical", "2,2,4Trimethylpentane"];
exChemicalGraph = MoleculeGraph[exChemical, IncludeHydrogens -> False]
```



Next, we need a way to enumerate the pairs of vertices to compute the pairwise distances.

Enumerate all pairs of vertices:

```
In[3]:= Subsets[VertexList[exChemicalGraph], {2}] // Shallow
Out[3]//Shallow=
{{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {2, 3}, {2, 4}, {2, 5}, <<18>> }
```

Now that we know how to enumerate the vertices, we can use **GraphDistance** to get the distance between any two vertices, and the Wiener index of the graph is the total of those distances.

Calculate the Wiener index:

```
In[4]:= wienerIndex[g_Graph] :=
  GraphDistance[g, ##] & @@@ Subsets[VertexList[g], {2}] // Total
```

Example:

```
In[5]:= wienerIndex[exChemicalGraph]
Out[5]= 66
```

A simple unbranched alkane is a special case: Its graph is called a **path graph**, and its Wiener index has an algebraically closed form. Since the algorithm to predict the boiling point involves computing the value of the unbranched variant first, it makes sense for us to use this formula on them because it is much faster than summing the vertex distances.

Derive the closed form for the Wiener index of a path graph:

```
In[6]:= Simplify[Sum[m (nCarb - m), {m, 1, nCarb}]]
Out[6]= 1/6 (-1 + nCarb) nCarb (1 + nCarb)
```

Define the new Wiener Index for a path graph:

```
In[7]:= wienerIndex[nCarb_Integer | nCarb_Real] :=  $\frac{1}{6}$  nCarb (nCarb - 1) (nCarb + 1)
```

A helper so that **WienerIndex** accepts any molecule-like **Heads**:

```
In[8]:= wienerIndex[mol_] := wienerIndex@MoleculeGraph[mol, IncludeHydrogens -> False]
```

Wiener also defined another topological index in the original paper, the **polarity number**<sup>[2]</sup>. It is the number of carbon pairs separated by three bonds. It is known to be related to the alkane's steric characteristics<sup>[3]</sup>.

Define the polarity number:

```
In[9]:= polarityNumber[g_Graph] := Count[
    GraphDistance[g, ##] & @@@ Subsets[VertexList[g], {2}],
    3
]
polarityNumber[mol_] :=
    polarityNumber@MoleculeGraph[mol, IncludeHydrogens -> False]
```

A path graph also has a trivial polarity number in a closed form.

Define polarity number for a path graph:

```
In[11]:= polarityNumber[nCarb_Integer | nCarb_Real] := Max[nCarb - 3, 0]
```

However, **GraphDistanceMatrix** can compute all the distances between vertices faster, so it is generally more efficient to calculate both indices with it at once, especially when using a lot of **Maps**. It computes the same values without having to create as many new lists.

Calculate two indices simultaneously:

```
In[12]:= wienerBothIndices[g_Graph] := With[{disMat = GraphDistanceMatrix[g]},
    {Total[disMat, 2] / 2, Count[disMat, 3, 2] / 2}
]
wienerBothIndices[mol_] :=
    wienerBothIndices@MoleculeGraph[mol, IncludeHydrogens -> False]
wienerBothIndices[nCarb_Integer | nCarb_Real] :=
    { $\frac{1}{6}$  nCarb (nCarb - 1) (nCarb + 1), Max[nCarb - 3, 0]}
```

Now that we can calculate those two indices, it is possible to predict the boiling point of an alkane with the method proposed by Wiener<sup>[2]</sup>. It involves two steps:

1. Predict the boiling point of the unbranched alkane of the same number of carbons with Egloff's equation. Egloff proposed this equation to show the trend that the boiling point increases with the number of carbons<sup>[1]</sup>:

$$T_{\text{boil}} \approx 745.42 \text{ K} \cdot \log_{10} (n_{\text{carbon}} + 4.4) - 416.24 \text{ K}.$$

Egloff's equation:

```
In[15]:= unbranchedBP[nCarb_] := 745.42 K * Log[10, nCarb + 4.4] - 416.25 K
```

2. Find the difference in the boiling temperatures. Wiener suggests that if two alkanes have the same number of carbons, the difference in their boiling points  $\Delta T$  is related to the difference in their respective Wiener indices  $\Delta w$  and path numbers  $\Delta p$ :

$$\Delta T \approx 98 \text{ K} \cdot \frac{\Delta w}{n_{\text{carbon}}^2} + 5.5 \text{ K} \Delta p$$

Boiling point predictor:

```
In[16]:= predictBP[mol_Molecule | mol_Entity] := Module[{nCarb, Δw, Δp},
  nCarb = AtomCount[mol, "C"];
  {Δw, Δp} = wienerBothIndices[mol] - wienerBothIndices[nCarb];
  unbranchedBP[nCarb] +  $\frac{98 \text{ K } \Delta w}{n_{\text{Carb}}^2}$  + 5.5 K Δp
]
```

From experiments, the prediction from this method is usually very close to the experimental value.

Deviations of three example predictions:

```
In[17]:= predictBP[#] - UnitConvert@#["BoilingPoint"] & /@
{ butane CHEMICAL, 2,3-dimethylhexane CHEMICAL, 3-methylhexane CHEMICAL }

Out[17]:= { 0.0762654 K, -0.779107 K, 0.938014 K }
```

## Finding Structure from Boiling Point

With a method to predict the temperature from the alkane's structure (the "forward" problem), now we can move on to our original problem: finding a set of alkanes with a specific boiling point. This problem is significantly less trivial than the "forward" problem because there is an infinite number of alkanes. However, it is possible to find a range of possible numbers of carbons in the molecule first and determine the structure of the alkane from the Wiener index.

### Prediction Based on Boiling Point

We know that the unbranched variant has the highest boiling point. Therefore, given a known boiling point, we can first determine the lowest possible number of carbons to have that boiling point (i.e. in the unbranched version), then generate some alkanes with that number of carbons or higher and see if they have a matching boiling point.

### Method 1: Brute-Force

#### Alkane Search

The first and the most obvious method to find a matching structure is by generating all possible molecules with a given number of atoms. Therefore, we want to create a function that generates all alkane isomers with a given number of carbon atoms.

One way to do this is by recursively growing an alkyl tree of carbon atoms. Each carbon atom is a node of the tree, and it connects to three child nodes. Suppose we need  $n$  carbons:

1. We treat one carbon as the root node and distribute the other  $(n - 1)$  carbons into three child nodes. This distribution is an integer partitioning problem, except that we allow zeroes.

Ways to distribute 4 carbon atoms:

```
In[1]:= IntegerPartitions[4 - 1, {3}, Range[0, 4 - 1]]
```

```
Out[1]= {{3, 0, 0}, {2, 1, 0}, {1, 1, 1}}
```

2. For each way to distribute those carbon atoms, we go back to Step 1 to generate child trees.
3. Finally, we combine all possible ways to create the child trees with **Outer** and get all possible alkyl trees with  $n$  carbons.

Since we are using a recursive approach, there might be instances where we need to run **generateAlkylTree** with the same parameters. Therefore, we wrap **Once** around the entire expression so that the results are cached and not recomputed.

Generate alkyls as a **Tree**:

```
In[2]:= generateAlkylTree[nCarb_Integer] := Once@Flatten@Table[With[
    {possibleSubBranches = generateAlkylTree /@ branching},
    Outer[Tree[Atom["C"], {##}] &, Sequence @@ possibleSubBranches]
], {branching, IntegerPartitions[nCarb - 1, {3}, Range[0, nCarb - 1]]}]
```

Starting condition:

```
In[3]:= generateAlkylTree[0] := {Tree[Atom["H"], {}]}
```


From there, it is easy to find the corresponding alkane by simply adding a hydrogen atom to it. We get some duplicates from **generateAlkylTree**, so we can delete them by removing isomorphic molecule graphs.

Convert the trees to molecules and remove duplicate ones:

```
In[4]:= generateAlkanes[nCarb_Integer] /; nCarb > 0 :=
    Molecule /@ DeleteDuplicatesBy[
        UndirectedGraph[TreeInsert[#, Atom["H"], {1}]] & /@
        generateAlkylTree[nCarb],
        CanonicalGraph
    ]
```

For example, to generate all the isomers of decane:

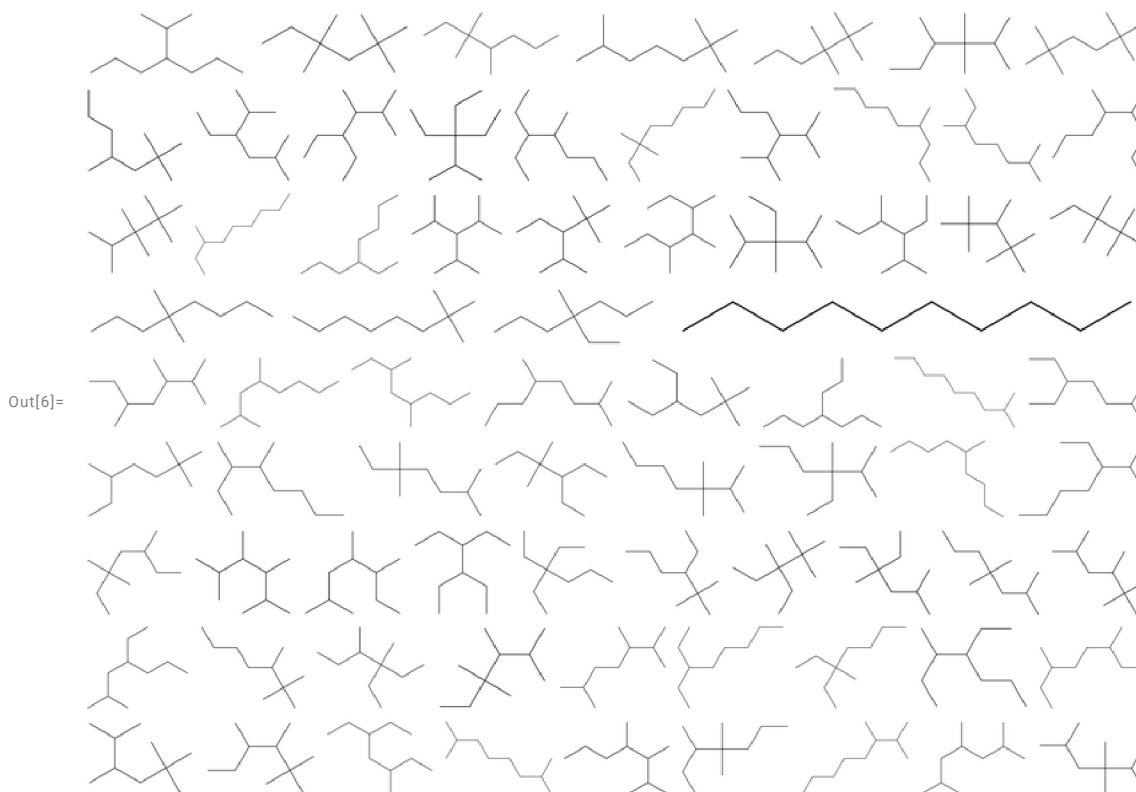
```
In[5]:= decanes = EchoTiming@generateAlkanes[10];
```

 2.05197

Fun fact: OEIS sequence A000602 shows the numbers of alkane isomers.

Show all decane isomers in images:

```
In[6]:= ImageCollage[MoleculePlot[decane, Style +], Automatic, 800]
```



## Understanding the Prediction

Now we have the functions to predict an alkane's boiling points. To successfully search molecules backwards, we first need to understand the range of numbers of carbons in which a boiling point is possible.

Generate some alkanes as examples:



```
In[1]:= isomers = generateAlkanes /@ Range[13];
```

### ■ Range of the Wiener index

First, the Wiener index is the highest when the alkane is completely unbranched; it decreases while the carbons branch and finally reaches a minimum when the carbon atoms are all packed around the center.

Calculate and show the maximum and minimum of the Wiener index:

```
In[2]:= Block[{indices = SortBy[{#, wienerIndex@#} & /@ Last[isomers], #[[2]] &]},
  Grid[{{"", "Molecule", "Wiener Index"},
    {"Max", Sequence @@ Last[indices]},
    {"Min", Sequence @@ First[indices]}}, Options +]]
```



	Molecule	Wiener Index
Max	Molecule [  Formula: C <sub>13</sub> H <sub>28</sub> Atoms: 41 Bonds: 40 ]	364
Min	Molecule [  Formula: C <sub>13</sub> H <sub>28</sub> Atoms: 41 Bonds: 40 ]	210

#### ■ Range of the polarity number

On the other hand, the unbranched alkane has a minimal polarity number because all atoms are the most spaced away.

Calculate and show the maximum and minimum of the polarity number:

```
In[3]:= Block[{indices = SortBy[{#, polarityNumber@#} & /@ Last[isomers], #[[2]] &]},
  Grid[{{"", "Molecule", "Polarity Number"},
    {"Max", Sequence @@ Last[indices]},
    {"Min", Sequence @@ First[indices]}}, Options +]]
```

	Molecule	Polarity Number
Max	Molecule [  Formula: C <sub>13</sub> H <sub>28</sub> Atoms: 41 Bonds: 40 ]	24
Min	Molecule [  Formula: C <sub>13</sub> H <sub>28</sub> Atoms: 41 Bonds: 40 ]	10

#### ■ Range of the boiling temperature prediction

Both the Wiener index and the polarity number generally increase linearly as the number of carbon increases. Since the coefficient of the Wiener index term is greater than that of the polarity number, the larger the former is, the higher the boiling temperature is predicted to be.

To better visualize how the boiling temperature prediction distributes, we can plot them in a graph.

Generate some boiling temperature predictions:

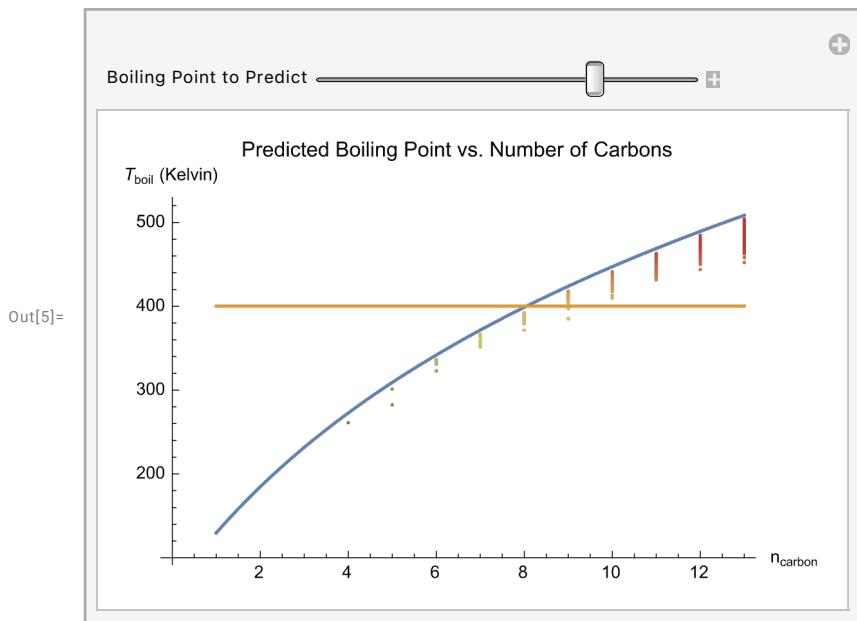
```
In[4]:= boilingPoints = QuantityArray@Map[predictBP, isomers, {2}];
```

Plot them:

```

In[5]:= Manipulate[Show[
  ListPlot[Flatten[MapIndexed[
    Table[Append[#2, bp], {bp, #1}] &, boilingPoints], 1], Options[+]],
  Plot[{QuantityMagnitude@unbranchedBP[nCarb], bp},
    {nCarb, 1, Length[boilingPoints]}],
  ], {{bp, 400, "Boiling Point to Predict"}, 100, 500}, SaveDefinitions -> True]

```



From the graph above, we can see that if we know the boiling point of an alkane, then there are generally a maximum of four possible numbers of carbons. Although this relationship does not necessarily hold for a higher number of carbons, we can assume so and proceed for the purpose of this project due to the lack of mathematical proof techniques.

## Putting Them Together

Now what we do is generate all possible isomers in that range of numbers of carbons and see which ones have the closest boiling point to the value we wanted. There are three steps to do this:

1. Find the approximate number of carbon atoms in the molecule from the boiling point with the reverse function of Egloff's equation.

**nearestNCarbons** finds the number of carbons with which an unbranched alkane has a boiling point near the given value. Since we know that the unbranched alkane has the highest boiling point, the unbranched alkane is the isomer with the lowest number of carbons with the given boiling point. I am using **Floor** instead of **Ceiling** here so as not to miss a possibility when the prediction is slightly lower than the given boiling point.

Get the approximate number of carbons given the boiling point:

```

In[1]:= nearestNCarbons[bPoint_Quantity] :=
  Floor[10 ^ (UnitConvert@bPoint + 416.25 K
    / 745.42 K - 4.4)]

```



2. Generate all alkane isomers of that number of carbons.

3. Return the closest matches.

Exhaustively search for an alkane:

```
In[2]:= findAlkanes[bPoint_Quantity] := Module[{
  nCarbsLow = nearestNCarbons[bPoint],
  nCarbsRange, possibilities
},
  nCarbsRange = Range[nCarbsLow, nCarbsLow + 4];
  possibilities = generateAlkanes /@ nCarbsRange // Flatten;
  SortBy[possibilities, Abs[predictBP[#] - UnitConvert[bPoint]] &]
]
```


Helper to take the first few matches from the results:

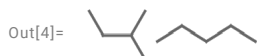
```
In[3]:= findAlkanes[bPoint_Quantity, nMax_Integer] := Take[findAlkanes[bPoint], nMax]
```

Now we can test the performance of this alkane finder!

Find alkanes with a boiling point of approximately 303 K and print the time taken:

```
In[4]:= (foundAlkanes1 = EchoTiming@findAlkanes[303 K, 2]) // MoleculePlot // Row
```

 0.360749



Get their real boiling points:


```
In[5]:= ToEntity[#]["BoilingPoint"] & /@ foundAlkanes1 // UnitConvert
```

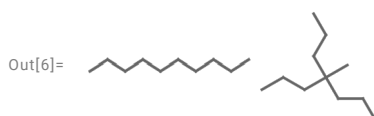
Out[5]= { 303.15 K , 309.21 K }

This function can also find larger molecules, although much slower.

Find molecules with a boiling point of approximately and print the time taken:

```
In[6]:= (foundAlkanes2 = EchoTiming@findAlkanes[174.2 °C, 2]) // MoleculePlot // Row
```

 107.566



Get their actual boiling points (not all molecules have their corresponding Entities because they may not exist in the real world):

```
In[7]:= Quiet@Check[ToEntity[#] ["BoilingPoint"],
  Echo[#, "Molecule does not exist as an Entity"],
  {ToEntity::noentp}] & /@ foundAlkanes2
```

» Molecule does not exist as an Entity Molecule [   Formula: C<sub>11</sub>H<sub>24</sub>  
Atoms: 35 Bonds: 34 ]

```
Out[7]= { 174. °C , Molecule [   Formula: C11H24  
Atoms: 35 Bonds: 34 ] }
```

We can see that this method works and gives us alkanes with boiling points pretty close to the given value. However, if the number of carbons is large, it takes too long because the program needs to generate and search all possible choices. It would be best to narrow down the search to a smaller range of possible alkanes.

## Method 2: Brute-Force with Direction

I came up with this idea based on Stephen Wolfram's two suggestions.

We already know from the previous section that isomers with the same number of carbons can have a range of boiling points. Egloff's equation is already a good enough predictor for the maxima (i.e. of the unbranched isomer), and we can fit a similar formula for the minima.

Get the minimum boiling points we predicted:

```
In[1]:= minima = Min /@ boilingPoints // QuantityMagnitude
```

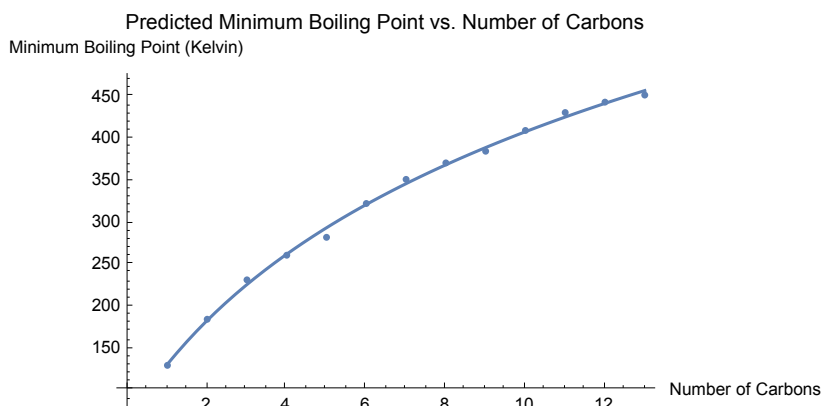
```
Out[1]= {129.691, 184.693, 231.693, 261.101, 282.459, 322.811,
  351.588, 371.246, 385.2, 409.977, 431.357, 443.777, 452.214}
```

Fit Egloff's log model:

```
In[2]:= minimaModel = NonlinearModelFit[
  minima, a Log[10, n + b] + c,
  {{a, 745.42}, {b, 4.4}, {c, -416.25}}, n
];
Normal[minimaModel]
```

```
Out[3]= -201.063 + 237.263 Log[3.05315 + n]
```

```
In[4]:= Show[Plot[minimaModel[nCarb], {nCarb, 1, 13}, Options[Options]], ListPlot[minima]]
```



Out[4]=

Convert the prediction result to a **Quantity**:

```
In[5]:= minimumBP[nCarb_] := Quantity[minimaModel[nCarb], "Kelvins"]
```

Each alkane will have a predicted boiling point that lies between the maximum and minimum. We define this relative position as the **relative boiling point**. Now, we propose a new assumption: since the relative boiling point depends on some properties of the structure, a chemical and one of its substructures are likely to have similar relative boiling points.

Define the relative boiling point:

```
In[6]:= relativeBP[nCarb_Integer, bPoint_] := Module[
  {
    minimum = minimumBP[nCarb], maximum = unbranchedBP[nCarb]},
  
$$\frac{bPoint - minimum}{maximum - minimum}$$

]
```

One that works with a **Molecule**:

```
In[7]:= relativeBP[mol_Molecule | mol_Entity] :=
  relativeBP[AtomCount[mol, "C"], predictBP[mol]]
```

Therefore, if we repeatedly add carbons to molecules and, at each step, we keep the few ones with a relative boiling point closest to our target and continue adding carbons to them, we might reach the target more efficiently.

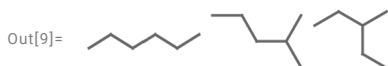
The first function adds carbon to all possible positions on an existing molecule (inspired by a Wolfram post<sup>[4]</sup>).

Find carbons with adjacent hydrogens and replace the hydrogen with carbon:

```
In[8]:= addCarbon[mol_Molecule] := MoleculeModify[
  mol,
  {"ReplaceSubstructureList",
   Atom["C", "HydrogenCount" → GreaterThan[0]] → "CC"}
]
```

Example:

```
In[9]:= MoleculePlot /@ addCarbon@Molecule["pentane"] // Row
```



It is also more convenient to have a carbon adder that works with a **List** of molecules.

**addCarbon** for a list of molecules:

```
In[10]:= addCarbonsList[mols_List] := DeleteDuplicatesBy[
  Flatten[addCarbon /@ mols],
  MoleculeValue[#, "CanonicalSMILES"] &
]
```

Now we define a function to filter the molecules and keep the closest few ones to our target.

Take molecules based on closeness to the target relative boiling point:

```
In[11]:= takeByRelativeBP[mols_List, nKeep_Integer, targetRelBP_] := TakeSmallestBy[
  mols,
  Abs[relativeBP[#] - targetRelBP] &,
  Min[nKeep, Length[mols]]
]
```

Now we put things together. We start from methane (which only has one carbon atom) and repeatedly add carbons to the molecule. After each step, we compute the relative boiling point and only keep the top **nKeep** ones that are closest to our target.


Find alkanes near a given boiling point with **nCarb** carbons:

```
In[12]:= findAlkanesFaster[bPoint_Quantity, nCarb_Integer, nKeep_Integer] := Module[
  {relBP = relativeBP[nCarb, bPoint]},
  Nest[
    takeByRelativeBP[addCarbonsList[#], nKeep, relBP] &,
    {Molecule["methane"]},
    nCarb - 1
  ]
]
```

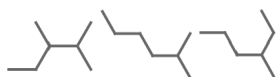
Let's test our new finder out!

Find alkanes with a boiling point of approximately 362 K and print the time taken:

```
In[13]:= (foundAlkanes3 = EchoTiming@findAlkanesFaster[362 K, 7, 3]) //
  MoleculePlot // Row
```

 0.09408

Out[13]=



Get their real boiling points:

```
In[14]:= ToEntity[#]["BoilingPoint"] & /@ foundAlkanes3 // UnitConvert
```

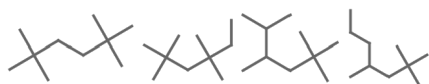
```
Out[14]= { 362.65 K , 355. K , 364.15 K }
```

Find alkanes with a boiling point of approximately 174.2° C and print the time taken:

```
In[15]:= (foundAlkanes4 = EchoTiming@findAlkanesFaster[174.2 °C , 10, 4]) //  
MoleculePlot // Row
```

```
0.243913
```

```
Out[15]=
```




Get their actual boiling points (not all molecules have their corresponding Entities because they may not exist in the real world):

```
In[16]:= Quiet@Check[ToEntity[#]["BoilingPoint"],  
Echo[#, "Molecule does not exist as an Entity"],  
{ToEntity::noentp}] & /@ foundAlkanes4
```

» Molecule does not exist as an Entity Molecule[ Formula: C<sub>10</sub>H<sub>22</sub>  
Atoms: 32 Bonds: 31 ]

```
Out[16]=
```

{ 137. °C , Molecule[ Formula: C<sub>10</sub>H<sub>22</sub>  
Atoms: 32 Bonds: 31 ] , 148. °C , 149. °C }

Method 2 finds alkanes much faster than Method 1. However, its quality quickly drops as the number of carbons increases.

## Conclusion

In this project, I explored the possibility of identifying alkanes based on a given boiling point using topological indices. It is delightful to see that the proposed methods, which search alkanes based on the Wiener index and Egloff's equation, can reach a reasonable accuracy within a range of different alkanes. This method could potentially assist in identifying unknown chemicals or helping people look for chemicals with specific properties.

However, due to my limited time and ability, there are also multiple areas worth further improvements and developments:

- While the proposed methods are effective, they are either slow or unreliable for higher boiling points. In the future, one could employ other methods, such as adjacency matrix generation, to further optimize the solution.
- This project only works with alkanes. In the future, it will be interesting to apply topology and graph theory to other families of chemicals and simplify the prediction of their physical properties.
- This solution uses the boiling point as the sole criterion for identifying chemicals. This framework can be more useful if it can to utilize other physical properties in the future.

- In this project, I made multiple simplifications and assumptions about the mathematical properties of the Wiener index. In the future, it is important to formally prove those claims and make this project more mathematically rigorous.

---

## Acknowledgements

First, this project would not have been possible without Stephen Wolfram's and Robert Nachbar's insights. Throughout the project, my mentor, Eryn Gillam, provided much support and assistance, and I would like to express my deepest gratitude to them. My TA, Yana Outkin, made this two-week journey much more interesting than I expected. Thank you! Finally, I am grateful to all my other teammates in my TA group and mentor group for your joy and support.

---

## References

- [1] G. Egloff, J. Sherman, and R. B. Dull, "Boiling point relationships among aliphatic hydrocarbons," *The Journal of Physical Chemistry*, vol. 44, no. 6, pp. 730–745, Jun. 1940, doi: 10.1021/j150402a006.
- [2] H. Wiener, "Structural Determination of Paraffin Boiling Points," *Journal of the American Chemical Society*, vol. 69, no. 1, pp. 17–20, Jan. 1947, doi: 10.1021/ja01193a005.
- [3] S. Nikolić, N. Trinajstić, and Z. Mihalić, "The Wiener Index: Development and Applications \*," *CROATICA CHEMICA ACTA CCACAA*, vol. 68, no. 1, pp. 105–129, 1995, Accessed: Jul. 14, 2022. [Online]. Available: <https://hrcak.srce.hr/file/260304>.
- [4] Wolfram, "Generate the Isomers of Decane," *Wolfram Language*, 2020. [Online]. Available: <https://www.wolfram.com/language/12/molecular-structure-and-computation/generate-the-isomers-of-decane.html.en>.