Reducing the Calculation Time of Knot Invariants

Yun-chi Tang

Summer 2021

1 Introduction

In knot theory, one problem of interest is to reduce the run time of calculating knot invariants such as the Jones' polynomial. One thing we can do is to seek to reduce the amount of information that we need to process through each time we add a crossing into the calculation by permuting the order of the crossings, which often is the only information needed to calculate knot invariants. To better explain this, let us define a permutation of a knot and the virtual width of a knot permutation:

Definition 1.1. *Given a knot K with n crossings $X_1, \ldots, X_n$, a permutation of a knot σ_K is a presentation of K such that crossings $X_{\sigma_K,1}, \ldots, X_{\sigma_K,n}$ are arranged in order from left to right and are $X_1, \ldots, X_n$ in some order.*

As shown in Fig. 1.1 as an example with Knot[9,13], to develop a permutation we will simply record the line segments leaving each crossing, arrange crossing in a desired order from left to right, and re-connect the line segments. Note from any permutation of the crossings we can develop it into many different permutations as there are many knot diagram presentations even given the location of the crossings.

Definition 1.2. *Given a knot K with n crossings, and a permutation σ_K of a knot, the virtual width at crossing $X_{\sigma_K,i}$ is the number of line segments connected to crossings $\{X_{\sigma_K,1}, \ldots, X_{\sigma_K,i}\}$ that connects to crossings $\{X_{\sigma_K,i+1}, \ldots, X_{\sigma_K,n}\}$. We will also refer to the virtual width as the v-width and the virtual width at $X_{\sigma_K,i}$ as $w_{\sigma_K,i}$. We define the v-width of the permutation to be the maximum of the v-width at all crossings minus $X_{\sigma_K,i}$.*

In reality this definition can be much simplified by a graphical definition: we notice that, if we allow for virtual crossings, then we can determine the width at a crossing by simply counting the number of points that a line directly to the right of the crossing, provided that all line segments directly go from the left to the right without circling other crossings. With that, we will define an alternate and equivalent definition of the virtual width:

Definition 1.3. *Given a knot K with n crossings, and a permutation σ_K of a knot, the v-width at a crossing is the number of points at which a vertical line directly to the right of the crossing will meet a virtual knot diagram with the crossings arranged in the order given by the permutation and the line segments not circling around crossings it is not connected to.*

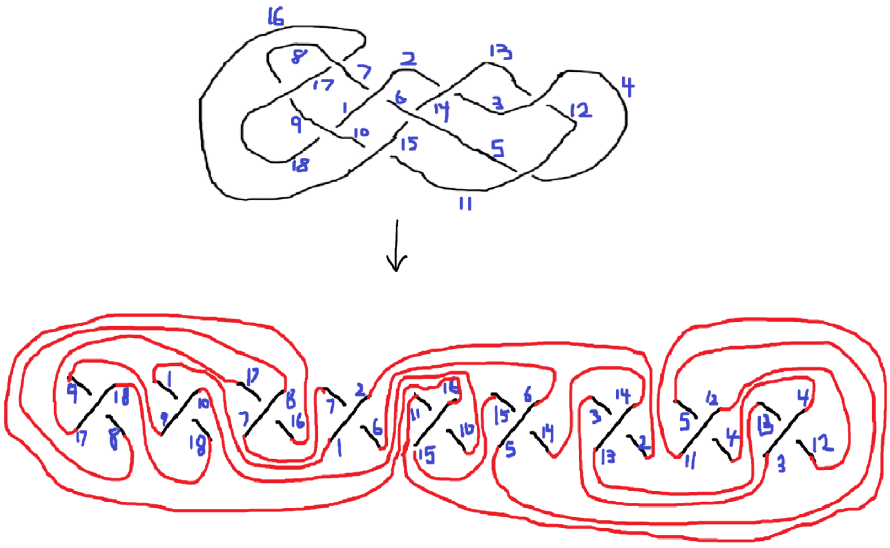

Figure 1.1: Developing A Knot Permutation

We can then graphically show what the virtual width of the same knot presentation given in Fig. 1.1 will be as shown in Fig. 1.2, and we recognize that the width of the knot permutation is 4.

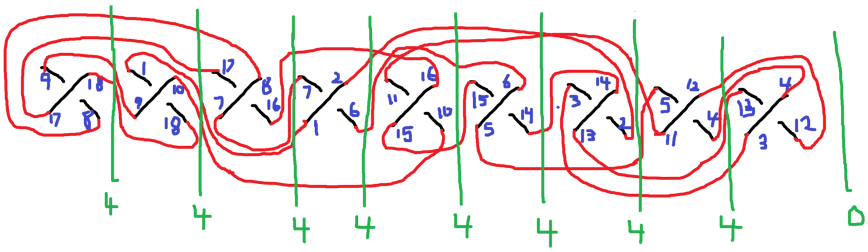

Figure 1.2: Developing A Knot Permutation

With that, there are two different methods that we can employ to permute the order of the crossings to reduce the information that we are required to process while calculating invariants, thus reducing the run time:

- Now we defined the definition of a permutation and v-width, we say a permutation m_K of a knot K is a minimum v-width permutation if all other permutations have v-width no less than the width of m_K and define its v-width to be the minimum v-width. We can seek to find the minimum v-width.
- Now we know the v-width at each crossing, we can establish a constant involving the v-width corresponding to the run time of a particular knot invariant calculation, and seek to reduce the constant.

In Section 2 we will seek to explore the first method, and in Section 3 we will seek to

explore the second method[1].

2 Minimum v-Width Knot Permutations

We can calculate a knot presentation's v-width from its PD notation as follows: We start by taking the first crossing listed, and we see the v-width at that crossing will be presented by the four line segments coming out the crossing. Afterwards, for each new crossing we add the edges connected to that crossing but not connected to prior crossings while deleting the edges connected to that crossing but connected to prior crossings, and with that we obtain the v-width at that crossing. Iterating through all possible permutations of crossings give a minimum v-width presentation.

For knots with a small number of crossings it will be easy to do; in fact we will be able to determine a minimum v-width permutation for all knots with the number of crossings between 3 and 10 inclusive[2]. One question that arises is: are we able to draw the presentations?

It turns out doing so can be difficult:

2.1 Difficulty in Drawing a Knot Permutation Presentation

To illustrate the difficulty in drawing a knot permutation presentation, let us look at the knot permutation shown in Fig. 1.1 and see how we could have computationally constructed the knot permutation.

We will start by constructing the crossings, as in Fig. 2.1:

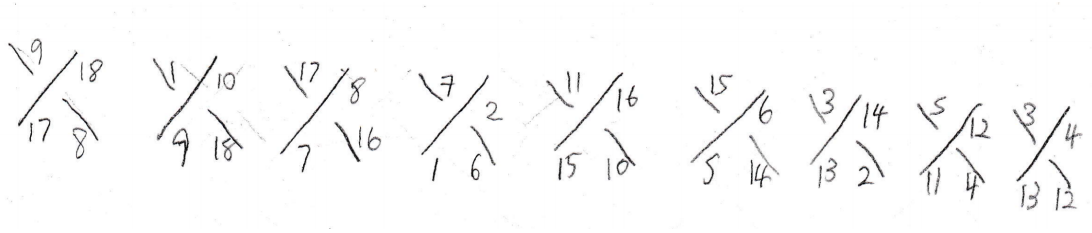

Figure 2.1: Constructing the Crossings

We first look at the second crossing and we recognize that we can connect line segments 9 and 18 so we will proceed to do that, as shown in Fig. 2.2.

Now we will try to connect the line segments without circling around any crossing other than the two crossings the line segment is connected to. We will define a line segment that will circling around crossings other than the two crossings the line segment is connected to as a circling segment. With that in mind, the line segment 10 will be rather straight forward to connect and we will connect it as shown in Fig. 2.3:

[1]This paper is supported by various programs written on Mathematica, and the files can be found on http://drorbn.net/AcademicPensieve/People/TangYC/index.html

[2]The program written to determine an order of the crossings leading to a minimum v-width permutation for all of these knots, as well as the minimum v-width of the knots, is in file MinVWidthList.nb

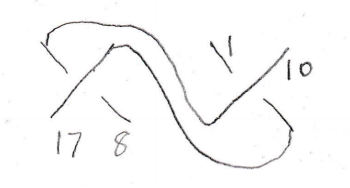

Figure 2.2: Connecting 9 and 18

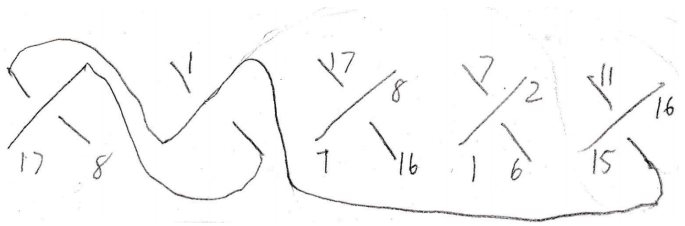

Figure 2.3: Connecting 10

Notice this line segment can go above the third crossing or below it. Depending on the decision we will, as illustrated by Fig. 2.4, determine how line segments 17 and 8 must connect, and subsequently 1 and 7:

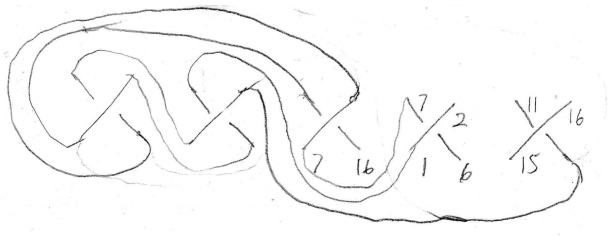

Figure 2.4: Connecting 17,8,1,7

At this point the process becomes more complicated - as shown in Fig. 2.5, line segment 16 will have to be a circling segment or it will intersect line segments $2, 6, 11, 15$. However, we could have connected the vertices in a different order and result in line segment 1 and 10 being the only circling segments of all line segments aforementioned. At this point, we notice that our decisions on which crossings will be circling crossings will affect how we position other line segments. For instance, the two described scenarios can chance the vertical order of line segments $2, 6, 11, 15$ coming out of the fifth crossing. Thus, it becomes difficult to determine how we select the order in which we connect these line segments.

Alternatively, we may argue that it might be easier to first connect line segments with the lowest number of crossings between the end crossings it is connected to, but that is precisely what we attempted to do in the first procedure. Following this procedure may result in unnecessary circling around crossings which complicate the process of trying to construct them.

To make matters even more complicated, we do not have it in this case, but if we want all four line segments attached to a crossing to go to the right of the crossing, there are four configurations in terms of the top-to-bottom order that the line segments leave the crossing as shown in Fig. 2.6.

These variations, which also applies to crossings where all four line segments attached to a crossing go the the left of the crossing, makes it complicated to obtain a good algorithm to draw the presentation of a minimum width permutation of a knot. Thus, we wonder if it is possible to obtain appropriate permutations such that the knot can be drawn in such a way that we avoid all the difficulties mentioned above?

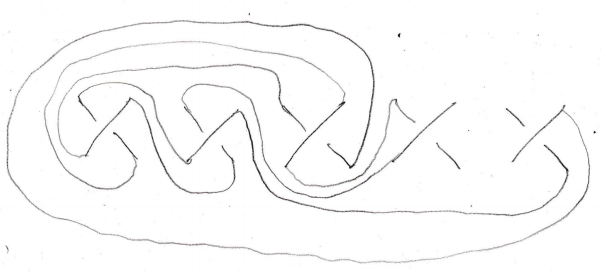

Figure 2.5: Circling 16 Around Other Crossings

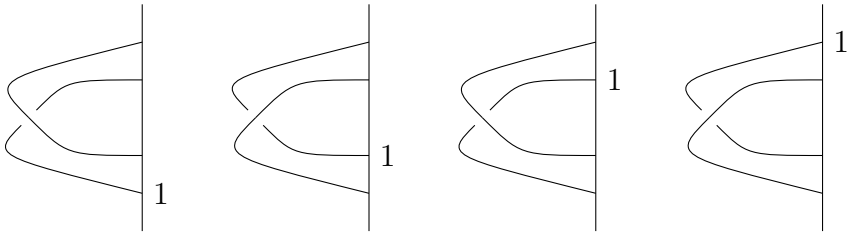

Figure 2.6: Four different ways the line segments can be arranged in top down order leaving the crossing; the "1" represent what we select as the first crossing in the PD notation

2.2 q-Width Presentation, Resulting Diagrams, and Discussion

To answer the question, we will define the following:

Definition 2.1. *Given a knot K with n crossings, and a permutation σ_K of a knot, the front $F_{\sigma_K,i}$ is defined as a vertical line directly to the right of $X_{\sigma_K,i}$ as well as the intersection points at which the line intersects K. We define the graphic width, which we will also call it the g-width, of $F_{\sigma_K,i}$ to be the number of intersection points on $F_{\sigma_K,i}$, and we will define the g-width of shelf permutation σ_K, $W[\sigma_K]$, to be the maximum of all $F_{\sigma_K,i}$ values, ie.*

$$W[\sigma_K] = \max_{i=1,2,\ldots,n} F_{\sigma_K,i}.$$

A knot presentation's g-width and v-width may not be equal. For example, going back to the knot used to show the virtual widths in Fig. 1.2, we recognize that the knot permutation shown in Fig. 1.1 will have graphical width 8 since the largest graphical width at a crossing is 8, as shown in Fig. 2.7.

Another question is whether the minimum v-width of a knot will be equal to the minimum g-width of a knot. However, this question is hard to answer as it is difficult to computationally define what a g-width of a knot presentation is until we construct the presentation. One way to solve this problem is to see if we can in certain permutations we can construct all the line segments without needing to worry about all the problems mentioned in Section 2.1, hence arriving at a point where we can determine the g-width easily.

With that in mind we will try to define such permutations and how to construct it. First, the following definition will more rigourously define the permutation and the computational procedure:

Definition 2.2. *Given a knot K with n crossings in permutation σ_K, we define a left half front of $X_{\sigma_K,i}$, $L_{\sigma_K,i}$, to be a vertical line in between the graphical front to the left of the*

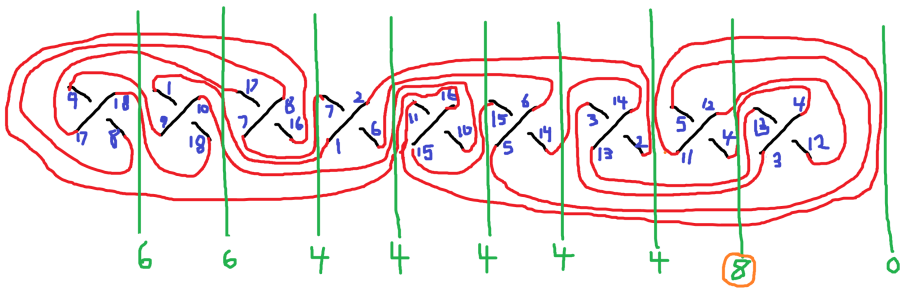

Figure 2.7: Developing A Knot Permutation

crossing and the intersection points with the strands directly extended from $X_{\sigma_K,i}$. *(If* $i = 1$, *the left half front is going to be a vertical line to the left of* $X_{\sigma_K,i}$ *with no points on it.) We can similarly define right half fronts.*

With that, let us rigourously define the type of permutation that we want:

Definition 2.3. *Given a knot* K *with* n *crossings, and permutation* σ_K *we claim that* σ_K *is a quick width permutation, which we will also call a q-width permutation, if* $L_{\sigma_K,1}$ *is the only left half front and* $R_{\sigma_K,n}$ *is the only right front with no intersection points, no line segment intersects a front twice, and we can connect the points on each front to half fronts immediately to the left and right without any edges intersecting.*

An example of a left half front, a right half front, and a front is shown in Fig. 2.8:

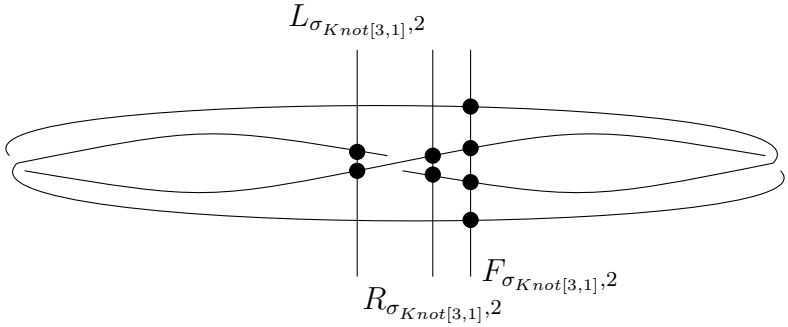

Figure 2.8: A Permutation of Knot[3,1], Front $F_{\sigma_{Knot[3,1]},i}$, and the Half Fronts Around It

We will also define what a minimum q-width permutation is:

Definition 2.4. *Given a knot* K *with* n *crossings we define the minimum q-width to be the minimum v-width across all q-width permutations. We will also define a permutation to be a minimum q-width permutation if its v-width is the minimum q-width.*

We notice that given a q-width permutation the v-width will be equal to the g-width as each edge will cross a given front either 0 or 1 times. The question now becomes how the minimum q-width compare to the minimum v-width and the minimum g-width.

Before that, though, we need to determine if a q-width permutation indeed exist for each knot with the number of crossings between 3 and 10 inclusive. We will constructively determine, given all permutations, if it is a q-width permutation via the following steps:

First, we establish all left half fronts by extending line segments to the line representing the left half front if the line segment is connected to a crossing on the left of the crossing where it is extended from, and similarly for the right half fronts, except the first and last crossing. If the permutation has more than one crossing with an empty left half front, or a crossing with an empty right half front, the process terminates.

We will start by looking at $F_{\sigma_K,1}$. From Fig. 2.6 there are four different combinations depending on how the line segments connected to $X_{\sigma_K,1}$ is arranged. For each of the combinations we extend the line segments to the line representing that front, and then extend the line segments to the line representing $R_{\sigma_K,1}$.

Then, as shown in Fig. 2.9 as an example, starting with what we will construct in the first two steps, we look at the line segments crossing through the line representing $L_{\sigma_K,2}$ and connect line segments from $R_{\sigma_K,1}$ to $L_{\sigma_K,2}$. If this cannot be done in such a way that no two line segments cross and no segments becomes circling segments, we terminate the process. Otherwise, we then extend the remaining unconnected lines all the way to $R_{\sigma_K,2}$.

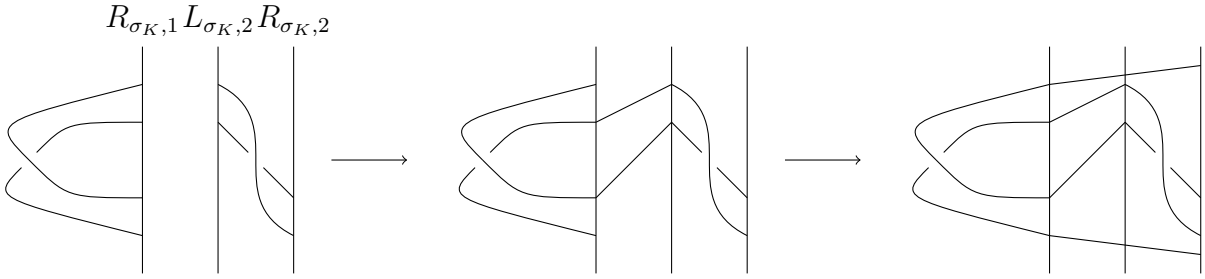

Figure 2.9: Illustration of a step of the quick width construction

We repeat this process until it is terminated or we reach the final crossing. At that point, we try to connect the remaining unconnected line segments to the line segments extending from the last crossing. If we are able to do so by any means without line segments crossing each other, we have constructed a quick width permutation. If not, we terminate the process.

It turns out all knots with the number of crossings being between 3 and 10 inclusive has a q-width permutation[3]. Through computation we notice, of all knots with the number of crossings being between 3 and 10 inclusive, there are six knots where the minimum q-width is not equal to the minimum v-width[4]. In all six of these cases, the minimum v-width is 4 and the minimum q-width is 6:

- Knot[9,18], PD Notation: PD[X[1, 4, 2, 5], X[3, 12, 4, 13], X[5, 14, 6, 15], X[9, 18, 10, 1], X[17, 6, 18, 7], X[7, 16, 8, 17], X[15, 8, 16, 9], X[13, 10, 14, 11], X[11, 2, 12, 3]]
- Knot[9,27], PD Notation: PD[X[1, 4, 2, 5], X[3, 10, 4, 11], X[11, 1, 12, 18], X[5, 13, 6, 12], X[13, 17, 14, 16], X[7, 14, 8, 15], X[15, 6, 16, 7], X[17, 9, 18, 8], X[9, 2, 10, 3]]

[3]This is true and the aforementioned procedures, as well as the crossing permutation of a minimum q-width permutation of each knot, is shown in MinQuickWidth.wl and MinQuickWidthData.wl respectively.

[4]File QuickWidth.nb contains details on what the minimum q-width for each knot is.

- Knot[10,24], PD Notation: PD[X[1, 4, 2, 5], X[11, 14, 12, 15], X[3, 13, 4, 12], X[13, 3, 14, 2], X[5, 16, 6, 17], X[9, 20, 10, 1], X[19, 6, 20, 7], X[7, 18, 8, 19], X[17, 8, 18, 9], X[15, 10, 16, 11]]
- Knot[10,25], PD Notation: PD[X[1, 4, 2, 5], X[5, 14, 6, 15], X[3, 13, 4, 12], X[13, 3, 14, 2], X[11, 20, 12, 1], X[19, 6, 20, 7], X[9, 18, 10, 19], X[7, 16, 8, 17], X[17, 8, 18, 9], X[15, 10, 16, 11]]
- Knot[10,42], PD Notation: PD[X[1, 4, 2, 5], X[3, 10, 4, 11], X[11, 1, 12, 20], X[5, 13, 6, 12], X[15, 18, 16, 19], X[13, 9, 14, 8], X[17, 7, 18, 6], X[7, 17, 8, 16], X[19, 14, 20, 15], X[9, 2, 10, 3]]
- Knot[10,44], PD Notation: PD[X[1, 4, 2, 5], X[5, 12, 6, 13], X[3, 11, 4, 10], X[11, 3, 12, 2], X[13, 20, 14, 1], X[9, 15, 10, 14], X[15, 18, 16, 19], X[7, 16, 8, 17], X[17, 8, 18, 9], X[19, 7, 20, 6]]

Now the question is if in these cases the minimum q-width will be equal to the minimum g-width. To answer that question, it is sufficient to determine whether it is possible for the minimum g-width to be 4 which the minimum q-width is 6. That, however, will prove to be impossible.

Suppose for the sake of contradiction we have a permutation where the g-width is 4. Then it must not be a q-width permutation or else the minimum q-width is at most 4. With that, since the g-width is 4 the only issue preventing the permutation from being a q-width permutation is if a line segment l intersected a front F twice; if there are more than one empty left half front then the g-width will be at least 6, and this holds true if there are more than one empty right half front. With that, let us trace this line segment l. It will have a left most point or a right most point by Rolle's Theorem. Without the loss of generality suppose l has a left most point. Then examine the area enclosed by l and F to the left of F. It cannot contain any crossing since otherwise we need a crossing with an empty left half-front and the g-width will be at least 6. Thus, we can pull the left-most point to the right of F without passing through any crossing. With that we have eliminated one line segments intersecting a front twice while adding no others since other line segments are unaffected. We repeat the process on all such segments and we will result in l not intersecting any front twice. Repeat this process and we obtain a q-width permutation where the g-width is at most what it was initially, meaning the minimum q-width will be at most 4. Contradiction.

With that in mind, we conclude that for the aforementioned six cases the minimum graphic width is 6 and thus for all knots with the number of crossings being between 3 and 10 inclusive the minimum quick width is equal to the minimum graphic width and in all except the six cases mentioned, the minimum width.

With that we will be able to construct the knot diagrams for all minimum quick/graphic width permutations of the all the aforementioned knots[5].

[5]The package MinQuickWidthPresentation.wl can be used to see a minimum q-width permutation presentation for each of these knots, too see all the permuation presentations at the same time, please go to MinWidthKnotGraphsGood.nb.

3 Optimization of Run Time Via Local Minimization

3.1 Estimation of Run Time of Knot Invariant Calculation

Suppose we have a knot of n crossings where n is approximately 100, and we want to find the Kauffman Bracket for the knot, which will in turn help us find the Jones' Polynomial. Then we can estimate the run time of the algorithm based on the v-width at each crossing. More specifically, let us look at the algorithm [1] that is used to calculate the Kauffman Bracket, which involves, as shown as Fig. 3.1, taking crossings one by one and calculating the terms of the Kauffman Bracket consisted of terms before the front corresponding to the crossing that we are adding.

```
FKB[pdList_] := Module[{p, t1, t2, t3, t4, B, d, KB, todo},
  SetAttributes[p, Orderless];
  KB = 1;
  todo = pdList;
  While[Length[todo] > 0,
   x = First[todo];
   todo = DeleteCases[todo, x];
   t1 = KB (x /. X[i_, j_, k_, l_] → A * p[i, j] * p[k, l] + B * p[i, l] * p[j, k]);
   t2 = Expand[t1];
   t3 = t2 //. {p[i_, j_] * p[j_, k_] → p[i, k]};
   t4 = t3 /. {p[i_, i_] → d, p[i_, j_]^2 → d};
   KB = Expand[t4 /. {B → 1 / A, d → -A^2 - 1 / A^2}];
  ];
  KB
 ]
```

Figure 3.1: The Algorithm For The Kauffman Bracket

We will notice that at each crossing $X_{\sigma_K,i}$, for each possible smoothing consisted of crossings that were to the left of this crossing, we will have to check through all the line segments that the $F_{\sigma_K,i}$ crossed to see if it connects with the new crossings that we are adding in. Suppose this new crossing is $X_{\sigma_K,k+1}$ where $k < n$. Then the number of terms that we expect to see appearing is at most $kw_{\sigma_K,k}$ since there are $w_{\sigma_K,k}$ segments the new crossing can be connected to and for every new crossings we multiply a term onto the existing terms. Also, the number of possible configurations will be equal to Catalan number $C_{w_{\sigma_K,k}/2}$ since it is restricted on how the line segments will connect below the front and there are w points to be connected.

With that, we can define the sum $S_{\sigma_K} = \sum_{k=1}^{n} k \cdot w_{\sigma_K,k} \cdot C_{w_{\sigma_K,k}/2}$ for permutation σ_K and we seek to determine if we can estimate the time it takes to calculate the Kauffman Bracket given this permutation by $T = t \cdot S_{\sigma_K}$ where t is some constant. As shown in Section 2, calculating the sum only requires us to know the width at each crossing which takes significantly less time to calculate than knot invariants such as the Kauffman Bracket. Thus, if we succeed in doing so, we will be able to use S_{σ_K} to estimate the effectiveness of algorithms that we develop in trying to reduce the time to calculate the Kauffman Bracket.

Let us see if S_{σ_K} indeed represents the run time it takes to calculate the Kauffman Bracket. To test that we will see, by looking at smaller sized knots, whether S_{σ_K} divided

by the run time is approximately the same for all knots. We will examine Dunfield's list of knots, a list of 1000 knots where the size of each knot increase by 1 starting at 3, and we look at the first 70 knots in the list. For each knot, we generate 1000 permutations of crossings via the greedy algorithm (at each step we will randomly select a crossing that will lead to the least v-width at this newly-selected crossing)[6] and select the one with the lowest v-width. Afterwards we will calculate the run time it takes to calculate the Kauffman bracket for each knot as well as the S_{σ_K} value for each knot, and we will divide the run times in seconds by the S_{σ_K} values. When we do that, we notice the ratios are rather constant as shown in Fig. 3.2[7].

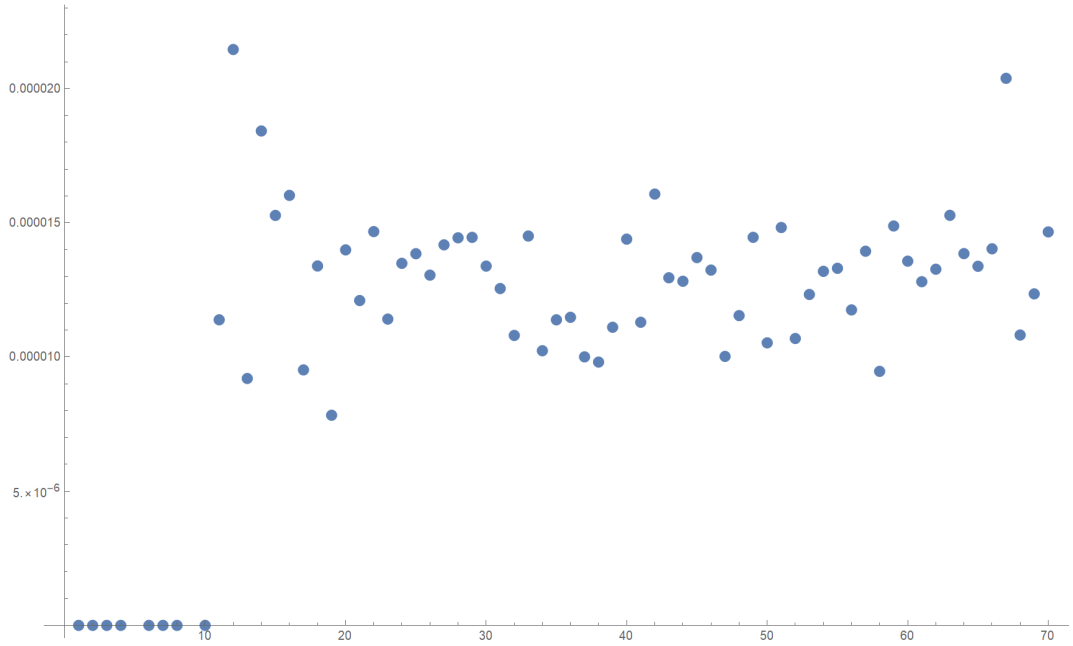

Figure 3.2: Ratios of Kauffman Bracket calculation times and S_{σ_K} values for 70 knots

From this graph we can also make an estimate of t: we will estimate t to be 0.000013 as the horizontal line with height 0.000013 is roughly in the middle of these data points. So, in developing ways to obtain thin representations of the knot we will make our estimation of run times based on the run time estimate $T = t \cdot S_{\sigma_K} = 0.000013 \cdot S_{\sigma_K}$.

3.2 Local Minimization

One tactic that we can employ is to reduce the S_{σ_K} value by looking at a small set of crossing every time and finding the best order of these crossings. What we can seek to accomplish is that we will, from left to right, take each consecutive set of x crossings in order, where x is a pre-determined number, and we will by brute force determine the best way to order these knots according to which permutation of these knots has the lowest resulting S_{σ_K} value, and change the order of the knots accordingly. We call this process the local minimization

[6]The source code of the greedy algorithm, as well as all other algorithms we will develop in this entire section, can be found in TestingFunction.nb.

[7]RunTimeConstant.nb contains how the procedure is implemented and how Fig. 3.2 is generated.

algorithm, which we will refer to as LM. A visualization of the algorithm with $x = 5$ is shown in Fig. 3.3.

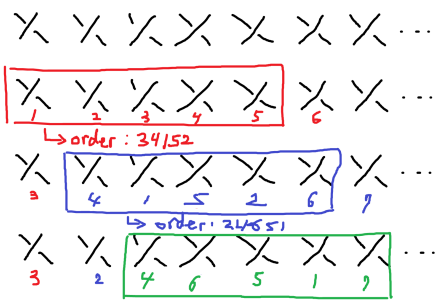

Figure 3.3: Visualization of the Local Minimization Algorithm

With that, let us look back at Dunfield's list of knots and we take the 98th knot, which has 100 crossings. Then, we can first generate 10 different permutations using the greedy algorithm and apply the local minimization algorithm to examine the S_{σ_K} as shown in Fig. 3.4[8].

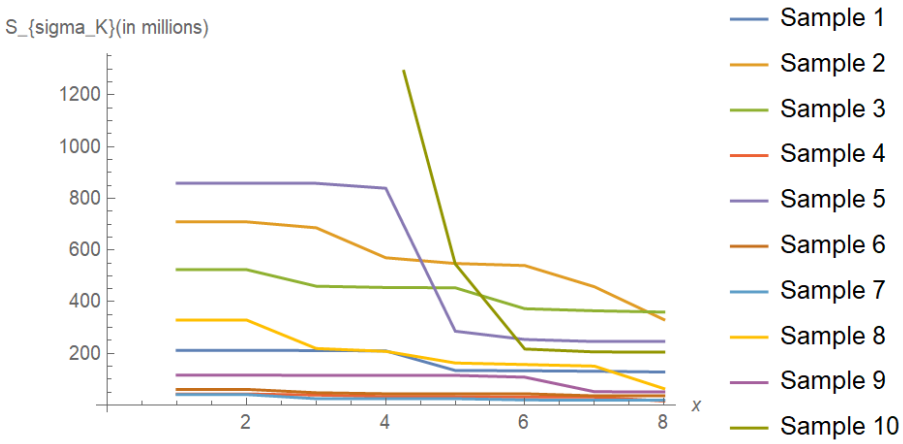

Figure 3.4: S_{σ_K} after local minimization with different x values

From this diagram we can make the following observations:

- Most of the drops in the sum value occurs when we adjust x to 5 or higher.
- It is extremely inconsistent in terms of when the sum value will drop and if it will drop at all. For instance, in sample 3, the initial sum value is not high but ended up not being effective at all when local optimization is applied. Similar problems happened for sample 7.
- Moreover, even if we apply these we cannot guarantee the result will be good. For instance in four of the ten cases shown in the graph the sum is at least 200 million,

[8]The code implemented to generated this diagram can be found in 4MethodsTesting.nb.

meaning the run time will be 2600 seconds which is a long time compared to scenarios that takes a lot less.

We notice that we can solve the last two problems with the two following steps:

- First we can generate many samples and select the best of them: for instance we will seek to generate 1000 different greedy algorithm-generated permutations, and determine the permutation with the least S_{σ_K} value, before we apply local optimization.
- If that is not sufficient, we can generate p sets of 1000 permutations and of those $1000p$ permutations, select the p with the least S_{σ_K} value, and then apply local optimization. The reason that we combine them instead of selecting 1 in each of the 1000 samples is that we will end up selecting samples with lower S_{σ_K} value and in less time.

Note that for the rest of this paper we will choose to ignore the second step since the extra time it will take to conduct this extra step will be too long; however, as it will be alluded to in Section 3.5 it might make sense to do so when the parameter gets large or in calculating other knot invariants.

Although repeating the process multiple times might further reduce the time it takes to calculate the Kauffman Bracket as it will enable the crossings originally at the back to be moved to the optimal positions, it often will not make sense to do so because the reduction in the time it takes to calculate the Kauffman Bracket is similar to, or even less than, the extra time it takes to conduct local minimization. In the example in Table 3.1, the estimated runtimes will serve to explain the effect: while linear optimization might serve to be useful, running linear optimization 3 times will prove to be useless. However, a question arises if we can perhaps reduce the time it takes to conduct local minimization in some way.

Function Performed	Avg. Est. RT After LM	Avg. Est. RT Doing LM
Initial	22.87	0
LM, x = 6	18.90	3.48
LM, x = 6, Repeat 3 times	18.21	10.44

Table 3.1: Comparison of Repeated Local Optimization

3.3 Partial Local Minimization

As it turns out, we can. One idea that we have is that in each step of the process in local optimization, if we can take a portion of sample without taking the set of all possible permutations, we might be able to make a significant enough reduction in much lesser time. This will enable us to repeat the process again and again while spending a similar amount of time as in Section 3.2. Of course, we will have to test that not taking the entire set of all possible permutations does not significantly increase the amount of time needed to calculate the Kauffman Bracket.

With that in mind, let us go back to the 10 permutations used in Fig. 3.4, and instead of taking the entire set of all possible permutations, we will take 10 percent of all possible

permutations and the current permutation for each time we seek to optimize the order of x crossings. The latter is selected to guarantee that we will never increase the time needed to calculate the Kauffman Bracket. Then, we will run the process for a total of 10 rounds, which will as a result take a similar amount of time as in Section 3.2. We will call this process a partial local minimization and refer to it as PLM. At this point we can observe from Fig. 3.5 that we will be able to achieve a better result than in Section 3.2, if we have equal x values, in the case of Fig. 3.5, $x = 6$ [9].

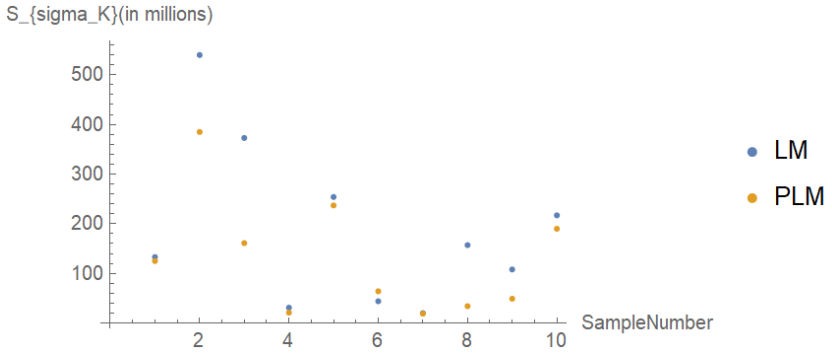

Figure 3.5: Comparison of Local Minimization and Partial Local Minimization

3.4 Reverse Local Minimization, Partial Reverse Local Minimization

Intuitively, if we go back to the summation formula for S_{σ_K} we notice that, given the same width, the further back the width is, the larger the S_{σ_K} value will be. With that in mind, one intuitive way that we can potentially further improve our results is if we start at the back of the knot and work our way forward, hence causing the heavier widths to be at the front in less steps.

We will apply this thinking to both the local minimization algorithm and the partial local minimization algorithm; we will start by looking at the last x crossings first, after that we will fix the last crossing and apply the process to the x crossings before that, and so on. We will call the methods reverse local minimization (referred to as RLM) and partial reverse local minimization (referred to as PRLM) First, we will apply it once again to the samples shown in Fig. 3.4 and we can see that although the difference seem to be close, the reverse partial local minimization could prove to be even better as a technique as partial local minimization as shown in Fig. 3.6 given a fixed x value, in this case $x = 6$ [10].

To fully verify whether conducting the local minimization algorithms is useful or not, we need a significantly larger data set and run more samples since the results appear to be close:

[9]The code implemented to generated this diagram can be found in 4MethodsTesting.nb.
[10]The code implemented to generated this diagram can be found in 4MethodsTesting.nb.

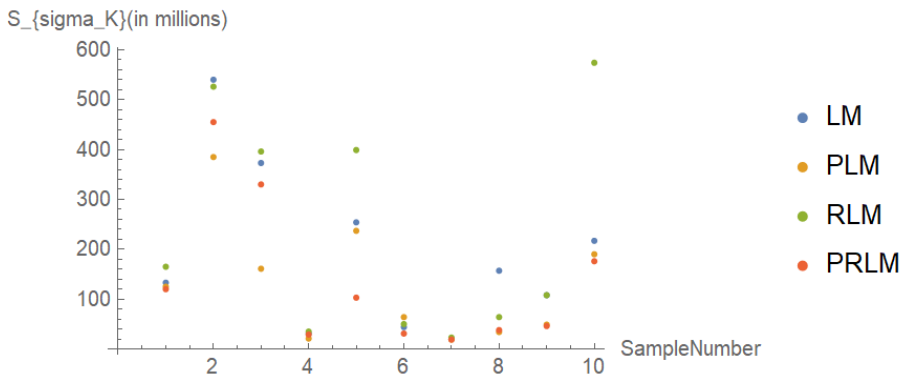

Figure 3.6: Comparison of Minimization Methods

3.5 Selection of Model and Parameters, Results

We will go back to the knot in Dunfield's list of knots with 100 crossings, and we will compare the four aforementioned methods by following the following steps for each model: We will first start by generating 50 different samples of 1000 permutations and select the best permutation with the lowest S_{σ_K} value. Then, we will conduct the specific algorithm to each selected permutation and record the run time. Finally, we will calculate the Kauffman Bracket and determine the time it takes to do so, and add it to the time it takes to conduct the minimization algorithm. The initial estimated time it will take to calculate the Kauffman Bracket will be approximately 24.43 seconds and as seen in Fig. 3.7, various levels of reduction is achieved[11].

Method	x = 6	x = 5
LM	24.19	21.17
PLM	22.97	23.20
RLM	22.44	20.87
PRLM	20.92	19.33

Table 3.2: Comparison of Methods, Size 100 knot

Sure enough, this estimation indicates that the reverse partial local minimization with $x = 5$ is the best method is going to be the best method for this. We will actually calculate the Kauffman Bracket and obtain the run time as shown in **??**.

However, we do notice that this might not apply for all knots. For example, if we increase the size of the knot to 200 by taking the 198th knot in the list, the estimated result will be much different, starting from the initial estimate of 193.76 seconds if we do not conduct any tests[12].

We notice in this case, reverse partial local minimization with $x = 6$ is the best method. In fact, in all four minimization algorithms, $x = 6$ will be better than $x = 5$ because the

[11] ModelRunTime.nb contains the full details of the estimated and actual run times of the minimization process and the Kauffman Bracket calculation times.

[12] ModelRunTime_Big.nb contains the full details of the estimated and actual run times of the minimization process and the Kauffman Bracket calculation times.

Method	x = 6	x = 5
LM	113.04	119.14
PLM	94.43	104.75
RLM	129.96	136.13
PRLM	88.36	93.70

Table 3.3: Comparison of Methods, Size 200 knot

advantage in terms of minimizing the needed run time will exceed the extra time it will take to conduct the local minimization methods. Also, we notice in this case we will reduce the run time by a much more significant margin, decreasing it by almost half compared to the approximately 20 percent decrease we got from the previous case.

The reason for both is explainable: we can notice the run time of all four developed algorithms, coupled with the greedy algorithm, increases quadratically as we increase the number of crossings as long as x is fixed. The run time of these algorithms increases linearly with the number of crossings as we simply go through the consecutive sets of crossings one by one from left to right while the run time of greedy algorithm increases quadratically with n. In fact it is very likely that, as the size of the knot continues to get larger, having larger x sizes will be more beneficial than $x = 5$ or $x = 6$.

To support this theory, let us take every 5th knot in Dunfield's list of knots, up to the 200th. Then we will estimate the total time it will take to, including the reverse partial local minimization run time, calculate the Kauffman Bracket of the knot after applying the reverse partial local minimization algorithm[13]. Then we will notice from Fig. 3.7 that when the size of the knot is less than 100, $x = 5$ is a better option than $x = 6$ and $x = 7$ [14]. After that, and especially if the size of the knot is greater than 150, $x = 6$ starts to become the better option and the reduction in estimated Kauffman Bracket calculation time will be significant compared to if we used other parameters.

We then take every 5th knot up from the 205th to the 400th, we will see from Fig. 3.8 that once the size of the knot approaches 250 (past the 10th knot in the list that we take in this step), $x = 7$ becomes a much better option, and that pattern becomes more exaggerated when the size of the knot goes beyond 300. With that, we confirm our theory that not only the minimization parameter will need to change as the knot gets larger, we also confirm that the effectiveness of the algorithm will likely become larger as the quadratic run-timed algorithm will have a greater effect on the estimated run time.

Finally, in this entire calculation we have set $p = 1$. However, it could be that if the knot gets larger it could be more advisable to have p larger than 1. This also could be relevant for other knot invariants we would like to calculate. However, since in our case each greedy algorithm-generated sample takes 6 seconds on the 100 crossing knot[15], we conclude that our knot is simply not big enough that increasing p will not make a difference.

[13]This time, since we have selected the minimization algorithm, we have compiled this into a package and we will use it. OptimizedKnotPresentation.wl is the package.

[14]All results in this section can be found in ParametersExperiment.nb.

[15]See SampleGenerationTest.nb on the procedure used to confirm the time needed to generate the sample.

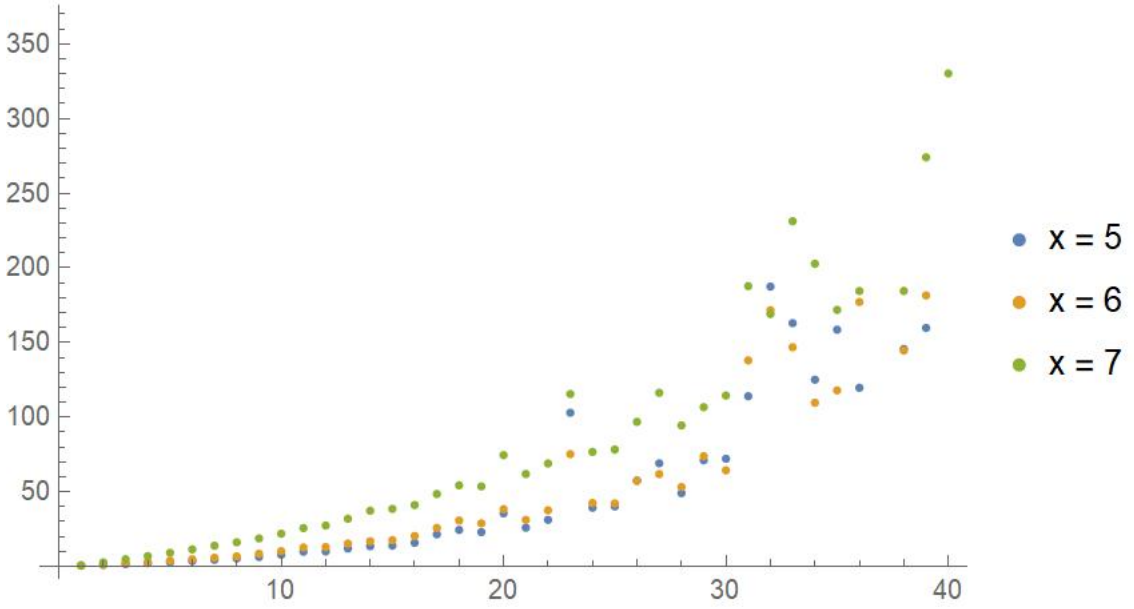

Figure 3.7: Comparison of Estimated Run Times Given x Values, For Every 5th Knot Up To The 200th

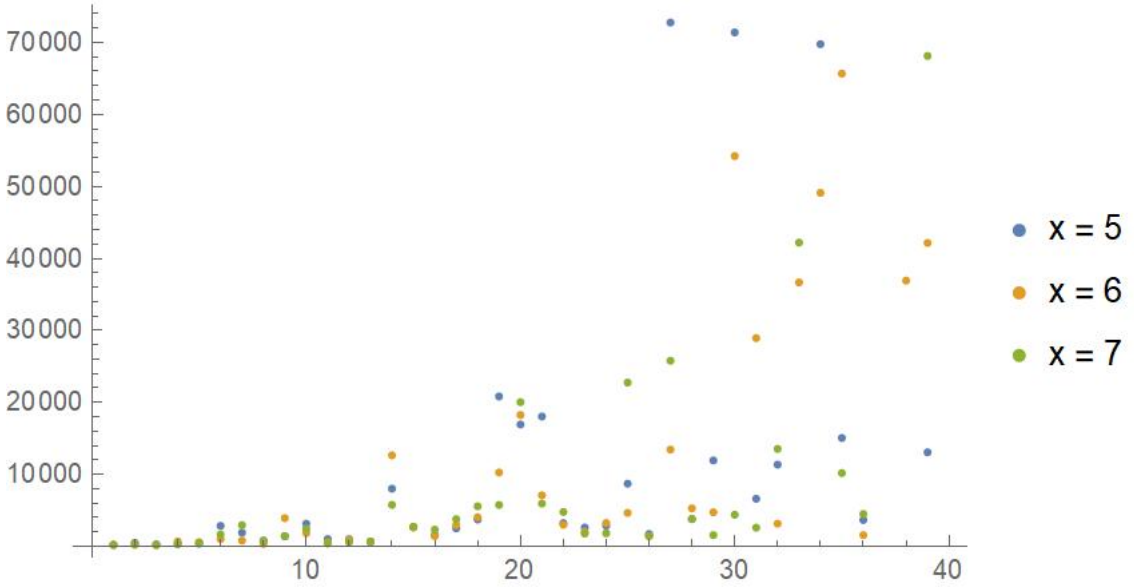

Figure 3.8: Comparison of Estimated Run Times Given x Values, For Every 5th Knot From The 200th To The 400th

References

[1] Bar-Natan, D. (2020) *FasterKauffmanBracket*. MAT1350F - Topics In Knot Theory. http://drorbn.net/AcademicPensieve/Classes/20-1350-KnotTheory/nb/FasterKauffmanBracket.pdf

[2] Dunfield. http://drorbn.net/AcademicPensieve/People/Dunfield/nmd_random_knots