2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9562113

An Artin Braid Group Representation of Knitting Machine State with Applications to Validation and Optimization of Fabrication Plans

Jenny Lin[1], James McCann[1]

Abstract— **Industrial knitting machines create fabric by manipulating loops held on hundreds of needles. A core problem in pattern making for these machines is transfer planning – coming up with a sequence of low-level operations that move loops to the appropriate needles so that knitting through those loops produces the correct final structure. Since each loop is connected to the larger piece in progress, transfer plans must account for not only loop position, but the way strands of yarn tangle around each other.**

We present the first complete, discrete representation of the machine's loop-tangling process. Our representation combines a braid from the Artin Braid Group with an array of explicit loop positions to fully capture loop crossings. By storing braids in the Symmetric Normal Form, states can be quickly compared and updated incrementally with machine operations. This representation can be used to verify the equivalence of transfer operations, providing an important tool in optimizing knit manufacturing.

We improve on prior work in transfer planning algorithms, which can only solve certain subclasses of problems and are frequently suboptimal in terms of fabrication time, by introducing a novel A* search heuristic and state-collapsing mechanism, which we show finds optimal transfer plans for a large benchmark set of small transfer planning problems.

I. Introduction

Industrial knitting machines are fast, flexible machines capable of creating a wide variety of shapes and structures. Not only can a single, continuous strand of yarn be used to form complex 3D surfaces, careful choice of knit stitches can produce various surface textures with drastically different appearances and mechanical properties [1]. Thus, knitting machines are used not only for technical garments like compression socks and athletic shoes, but also architectural formwork [2, 3] and large-scale installations [4, 5].

Programming for knitting machines can be difficult, as it involves carefully arranging low level operations, a task akin to programming in assembly language. Programmers must develop instructions that produce the desired output object and, ideally, do so as quickly as possible given the available machine resources. A key operation used by knitting machines to create more complex shapes and structures is the *transfer*, which moves loops around the machine. Transfers are used to re-arrange loops in order to set them up for subsequent knitting operations. However, many different sequences of transfer operations may achieve the same result with dramatically different fabrication time and stress on the yarn; a good knit programmer must make an informed choice among these sequences.

*This work was partially supported by Shima Seiki.

[1]Jenny Lin and James McCann are with Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA {jennylin, jmccann}@cs.cmu.edu

Fig. 1: The arrangement of loops on a knitting machine (left) can be represented as a braid (right) by ordering the needles and collapsing each loop into a single strand.

Furthermore, when checking that any two transfer sequences produce the same result, it is insufficient to check that all loops end up on the correct needles. Each loop is connected to the partially knitted object, so rearranging them can result in tangles, both desired and undesired.

In this paper, we address the problem of transfer planning by representing knitting machine states as mathematical braids to fully capture the loop positions and interlacing. Figure 1 shows an example of one such intermediate state and its corresponding braid. This discrete representation avoids the need for yarn simulation or geometric computations to track interlacing. State updates can be done in $O(n \log n)$ operations, where n is the number of loops in the problem, and state comparison in $O(m+n)$ operations, where m is the length of the underlying braid word (approximately proportional to the complexity of the yarn tangling). Given such a representation, it is natural to ask whether a discrete search can be used to find optimal transfer sequences.

We answer this question in the affirmative, presenting an A* search built on our braid-based state representation. We define the machine's transfer process in the context of transitions in a planning problem and present several heuristics that emerge from the braid-based representation's structure. In addition, we present a method to reduce the search space by converting equivalent states to a single canonical form. Our search returns solutions for transfer problems existing transfer planning algorithms [6, 7] cannot address and demonstrates that these algorithms frequently return suboptimal plans.

II. Related Work

The pipeline for transfer sequence design used by commercial systems [8] typically involves composing pre-planned sub-sequences of transfers which solve commonly-encountered situations (e.g., "cross these two loops", "move

this loop two needles to the right"). For more complicated cases, or in cases where the provided operations are not time-optimal or violate physical constraints of the object being fabricated, the user is left to come up with a custom plan using basic transfer operations. Confirming the correctness of the transfer sequence then requires either fabricating the pattern or running a physical simulation of the operations. Afterwards, it is up to the user to visually inspect the result for correctness, a task that becomes more difficult as the complexity of the problem increases.

In the domain of automatic and semiautomatic machine knitting patterns, tools either constrain the pattern space to problems which can be solved by existing general algorithms [9, 10, 11, 12], or they allow for custom plans that can be inserted to handle custom cases [13, 14]. While these methods can provide guarantees on the correctness of the result, there is no general method for merging plans that provides any bounds on final fabrication time.

The two primary algorithms that provide general solutions to subsets of the transfer planning problem space are collapse-shift-expand [6] and schoolbus+sliders [7]. Collapse-shift-expand can widen, narrow, and rotate tubes, and it guarantees that all loops arrive on the correct needle. However, when multiple loops have the same destination needle, it provides no guarantee about what order the loops will stack on the needle. It also cannot handle cables, which are loops that cross each other. Schoolbus+sliders is a simple algorithm that produces faster plans than collapse-shift-expand, but it only works on flat sheets. It can guarantee which loop is in the front when creating stacks, but cannot guarantee total stack order or solve cables.

III. Background

We start by providing a description of an industrial knitting machine and the operations relevant to transfer planning. We then provide a high level introduction to the Artin braid group and the symmetric normal form. For a more detailed description of braids, we recommend [15, 16].

A. Knitting Machine Model

An industrial v-bed knitting machine consists of two facing beds (rows) of hook-shaped needles, each of which can hold some number of loops. The second bed is what allows v-bed machines to construct not only sheets, but also tubes, which are flattened to lie on both beds. Each needle is labelled by the bed it belongs to (f for front and b for back) and its integer index within its bed. The machine may transfer all loops held on a needle to the needle across from it on the opposing bed. If we associate with each needle, n, a list of loops, $\mathsf{loops}(n)$, it holds ordered from tip to base, the transfer function can be defined as follows:

Definition 1 (Transfer Operation)**.** *A transfer between two needles* $\mathsf{xfer}(n,n')$ *represents the operation:*

$$\begin{aligned} \mathsf{loops}(n') &\leftarrow \mathsf{loops}(n') + \mathsf{reverse}(\mathsf{loops}(n)) \\ \mathsf{loops}(n) &\leftarrow [\,] \end{aligned}$$

The knitting machine may perform a transfer operation only between two needles that are aligned with each other. It can change needle alignment by using the rack operation, which slides the beds laterally. At a racking value r, front bed needle f_n is across from back bed needle b_{n+r}. The machine's possible racking values are constrained to some range $[R_{min}, R_{max}]$ (usually about $[-5,5]$ needles).

A transfer operation is performed when a mechanical part called the carriage passes over a needle and actuates it. Multiple transfers performed at the same racking can be grouped into a single *transfer pass*. The amount of time required for a transfer pass is independent of the number of operations therein, and significantly longer than the racking time. Thus, the number of transfer passes serves as a reasonable proxy for the overall time of a transfer plan.

Given a sequence of transfer operations, the number of transfer passes required is equivalent to the number of nonempty blocks of sequential transfers at the same racking value. Note that should the operations $\mathsf{xfer}(n,n')$ and $\mathsf{xfer}(n',n)$ occur in the same pass, from Definition 1 we see that they are equivalent to the single operation $\mathsf{xfer}(n',n)$.

While there are more complicated machines that employ additional beds or specialized holding locations for each needle, the v-bed model can emulate these machines using a technique called half-gauging, where (e.g.) even needles are used as temporary holding locations.

B. Artin Braid Group

Braid theory is the topological study of groups whose elements are intertwining strands with fixed endpoints [17]. More specifically, imagine a box with n starting points on the bottom and n ending points on top. Each start point has a strand that connects with some end point, with no two start points sharing the same end point. In addition, when following a strand from start to finish, one should only move upwards; if the strand 'turns around' and heads down, it is not a braid. The algebraic definition is as follows:

Definition 2 (The Artin Braid Group)**.** *The Artin Braid Group* B_n *on* $n > 1$ *strands is the group generated by generators* $\sigma_1^+ \ldots \sigma_{n-1}^+$ *with the equivalence relations:*

$$\begin{aligned} \sigma_i^+ \sigma_j^+ &= \sigma_j^+ \sigma_i^+ && \text{for } |i-j| > 1 \\ \sigma_i^+ \sigma_j^+ \sigma_i^+ &= \sigma_j^+ \sigma_i^+ \sigma_j^+ && \text{for } |i-j| = 1 \end{aligned}$$

When writing and drawing braids, we use the convention that positive crossing, σ_i^+, represents the ith strand crossing *over* the $i+1$st strand, and the inverse (negative) crossing, σ_i^-, represents the ith strand crossing *under* the $i+1$st strand. Braid words are products of generators, where the leftmost generator is the most recently executed generator, and the rightmost generator is least recent. An inverse of a word can be quickly found as follows:

Definition 3 (Braid word inverse)**.** *Given the braid word* $W = \sigma_i^{\pm} \sigma_j^{\pm} \ldots \sigma_k^{\pm}$*, the word* $W^{-1} = \sigma_k^{\mp} \ldots \sigma_j^{\mp} \sigma_i^{\mp}$ *is its inverse, where the product* $W^{-1}W$ *is equivalent to the trivial identity braid with no crossings,* ε*.*

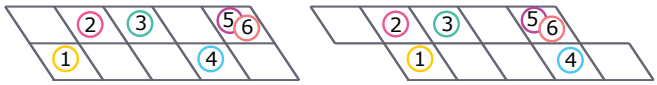

Fig. 2: The left-to-right, back-to-front ordering on loops (colored circles) on a machine at zero racking (left) and -1 racking (right). Note how the relative order of loops on the same bed does not change. This includes loops on the same needle.

Due to the equivalence relations, each member y of the braid group B_n is an equivalence class of braid words, all of which represent the same underlying topological braid. The word problem on braids asks whether two braid words W and W' belong to the same equivalence class, i.e. are the same topological braid. There exist a variety of solutions to the word problem, of which several use what are known as simple positive braids:

Definition 4 (Simple positive braid)**.** *The simple positive braids are the set of braids where all crossings are positive, and every pair of strands crosses at most once.*

The symmetric normal form [15] rearranges each braid word into a carefully ordered sequence of simple positive braids and inverse simple positive braids. This sequence is proven to be unique for each member of the braid group, thereby reducing the word problem to a question of strict equality.

IV. State Representation

In order to fully capture a knitting machine's state, we combine explicit loop locations with the Artin braid group, which can represent tangles between knit loops. Recall that Definition 2 requires a strict ordering on its strands. We choose to order the loops left-to-right, back-to-front, as seen in Figure 2 (note how the assignment of loops 1 and 2 swaps).

Observe that any transfers that occur at racking r do not change the loop ordering. It is only when the machine's racking value changes that any loops in back bed b would be sent to different needles in front bed f, potentially changing the loop ordering. Furthermore, loops on the same bed never change order relative to each other; a loop in f can only potentially cross a loop in b. Thus the resulting braid from a racking operation can be found by tracking the differences in loop ordering using Algorithm 1.

Let V be the braid word produced by a single racking operation and Y be the symmetric normal braid representing the previous state's ordering. The braid word VY represents the new state. Note that a single rack operation causes any two strands to cross only at most once, and resulting crossings are either all positive or all negative. Thus V is either a simple positive braid or its inverse. Therefore, the symmetric normal form of VY can be calculated in $O(n\log(n))$ operations [16].

Given an initial list of loop locations and an initial braid, the resulting state from any list of transfer operations can be determined. Because the ordering used for the braid word also captures the ordering of loops in a stack, it is sufficient for a state to only store each loop's needle position instead of explicitly storing the loop list of each needle.

Algorithm 1 Racking Operation

Input: ordered list of loops L, old racking r, new racking r'
Output: braid word V

$V \leftarrow \varepsilon$
if $r' < r$ **then**
  for $i = 0 \ldots n$ **do**
    if $L[i] \in b$ **then**
      for $j = i-1 \ldots 0$ **do**
        if $L[j] \in f \wedge L[i] + r' \leq L[j]$ **then**
          $V \leftarrow \sigma_j V$
else if $r' > r$ **then**
  for $i = n \ldots 0$ **do**
    if $L[i] \in b$ **then**
      for $j = i+1 \ldots n$ **do**
        if $L[j] \in f \wedge L[i] + r' > L[j]$ **then**
          $V \leftarrow \sigma_{j-1}^{-} V$
return V

V. Optimal A* search

Given this discrete representation of the knitting machine state, we now use search techniques to find minimum-length transfer plans. We base our search on A*, where a state's immediate neighbors are those reachable via any number of transfers followed by a single racking operation, and the goal has a braid equivalent to input braid W and loops at target needle locations L. We now model the constraints of the machine knitting process to restrict the search, and propose several modifications to the state representation to improve search performance.

A. Constraints

Loops on a knitting machine are constructed using a single continuous strand of yarn, where the amount of yarn between adjacent loops can be varied. This physical connection can break when connected loops are moved too far apart. To account for this, we define a slack constraint $[s_-, s_+]$ on the distance between connected loops l_a and l_b. In other words:

Definition 5 (Slack Constraint)**.** *Connected loops l_a and l_b respect slack when $s_- \leq pos(l_a) - pos(l_b) \leq s_+$, where*

$$pos(l) ::= \begin{cases} i & \text{if } l \in f_i \\ i+r & \text{if } l \in b_i \end{cases}$$

Note that, for loops on opposite beds, the machine's racking affects whether the loops respect slack. Thus the set of all connected loops which lie on opposite beds can be used to define a valid racking range $[V_{min}, V_{max}] \subseteq [R_{min}, R_{max}]$.

We also explicitly define which needles are available, as certain needles on the machine may be occupied by loops that should not be moved by the transfer plan. This must also be taken into account when determining whether a transfer is reversible.

B. Cost Model

Recall that the machine can execute any number of transfers in a single transfer pass as long as they occur at the same racking. Therefore, it's useful to think of transitions as $(\{xfer\}, rack)$ pairs, where $\{xfer\}$ is one of the 2^n subsets of n total distinct transfer operations for a given state, and *rack* is one of the valid racking operations for the state post $\{xfer\}$. The cost of a transition is 0 if $xfer$ is empty, and 1 otherwise.

C. State Equivalence

A* search stores visited states in memory in order to avoid expanding the same state multiple times. This makes fast state equality checks essential. In fact, we can do better than just strict equality. For two different states which produce the exact same set of subsequent states under expansion, we can prune the state space by searching only one of them. In this section we define a function, *canonicalize*, to identify such equivalence classes in which this property holds true.

Recall that $\mathsf{xfer}(n,n')$ can be reversed by $\mathsf{xfer}(n',n)$ if n' is an empty needle. Consider two states S and S', which are identical except for a single loop that is on needle n in state S, and needle n' in state S'. If $\mathsf{xfer}(n,n')$ is reversible and the transfer set from S to some state includes $\mathsf{xfer}(n,n')$, then that transfer can be reversed with $\mathsf{xfer}(n',n)$ to acquire the transfer set that would reach the same state from S'. If the transfer set from S does not include $\mathsf{xfer}(n,n')$, then S' can use the same set plus $\mathsf{xfer}(n',n)$. This means that in a search, no matter which of the two states are expanded first, it will visit all states that can be reached from the other state, making the second expansion redundant.

Now consider some non-empty transfer set X at racking r. This set would have cost 1. Any number of additional transfers can be performed before and after X, and as long as they also occur at racking r, they can be rolled into the existing transfer pass for cost 0. Thus if we define some reversible transfer function that sends all equivalent loop states to a single, canonical loop state, the function can be applied before the duplicate check, making it a simple equality check. These reversible transfers would then combine with the $xfer$ portion of the expansion, making it a zero-cost transformation. Let *canonicalize* denote an operation on a given state, which performs all reversible transfers on back bed loops, essentially loading as many loops as possible to the front bed without performing any irreversible transfers (Figure 3). Then, during the search, our code stores visited states and checks against them all under application of *canonicalize*.

D. Heuristics

To guide the search, we provide the following heuristics based on the braid word and the loop locations.

1) Braid Word Length: Let $len(Y)$ give the number of simple braids contained in the symmetric normal form Y. This notion of braid length changes by at most one after multiplication by another simple braid [16]. Rather than start with the identity braid ε and check equivalence against the target braid W, we can let the initial state be W^{-1} and let ε be the target braid. The resulting braid produced by the plan will still be the target braid W, and $len(Y)$, where Y is the symmetric canonical braid stored in the state, will be a consistent heuristic.

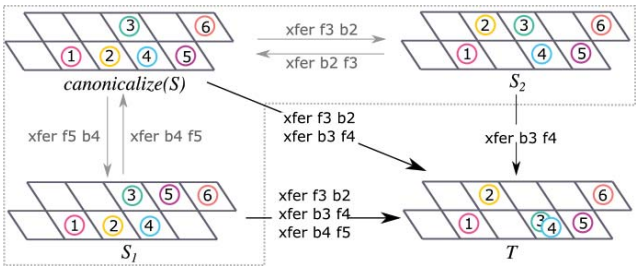

Fig. 3: A transition from S_i to state T costs a single transfer pass even if it is preceded by some number of reversible transfers. Thus the states are equivalent, and we can use a single canonical state $canonicalize(S)$ when representing them in a search.

2) Offsets: If we consider a relaxed version of the transfer planning problem that has infinite slack, reversible loop stacking, and is only concerned with needle position and not relative ordering, then a solution can be found by solely considering each loop's offset, or the single racking value that would move a loop from its current location to its destination. We can define $\mathscr{O}_{p,r}$ – the set of sets of offsets that can be brought to zero in p steps starting at racking r – using the following recurrence:

$$\mathscr{O}_{p+1,r'} \equiv \left\{\{x, x+(r'-r) \mid x \in S\} \mid S \in \mathscr{O}_{p,r}\right\} \tag{1}$$

$$\mathscr{O}_{0,r} \equiv \begin{cases} \{\{0\}\} & \text{if } r = 0 \\ \emptyset & \text{otherwise} \end{cases} \tag{2}$$

Notice that the number of offsets at most doubles every step. Thus we can establish the following lower bound:

Theorem 1. *Any problem with n unique non-zero offsets requires are least* $\lfloor log_2(n+1) \rfloor$ *passes to solve.*

Our offset table heuristic goes further, computing and storing $\mathscr{O}_{p,r}$ (effectively, a pattern database [18]) for $r \in [-8,8]$ and $p \leq 8$. In order to facilitate fast lookups in $\mathscr{O}$, our code builds a table – for every racking r – of all maximal[1] achievable offset-sets and their associated minimum step count. This table is sorted by step count. To look up a query set in the table for the current racking value, the code examines entries in order until a containing set is found and returns the associated step count. This lookup is accelerated by maintaining a "skip" value for each offset alongside each row in the table, indicating the next table entry in which that offset appears. These skip values are used by our code to avoid needing to check every row.

VI. Results

All experiments were performed on a mid-range workstation-class computer running Debian GNU/Linux with

[1]I.e., if both $S, R \in \mathscr{O}_{p,r}$ and $S \subset R$, then only R is stored.

an Intel Core i7-8700K 3.7GHz / 12-thread CPU – though our search is single-threaded – and 64GB of RAM.

a) Optimal plans: We compare the optimal plans produced by canonical-node A* search against two existing transfer planning algorithms: schoolbus+sliders (`sb+s`) and collapse-shift-expand (`cse`). Both algorithms have limitations on the types of problems they can and are best suited to solve, so we used three test sets for comparison.

For each test case, our test harness first ran the existing algorithm to generate a transfer sequence, then used that transfer sequence to construct a target state, and, finally, ran our search algorithm with that target state. (This procedure is needed because `cse` chooses stacking and rotation direction without user control, so it may plan to one of several possible output states.)

The first set, `flat-lace`, consists of all 28,696 unique transfer problems with eight loops, stacks of at most three loops, distance between loops increasing by at most one, and no loop crossings (cables/twists). Problems that only differ by translation are considered equivalent. In other words, this is the set of eight-loop problems that `sb+s` can solve. Results are shown in Figure 4. As expected, `sb+s` is able to solve these cases in relatively few (< 15) passes, and, indeed, is optimal in 2942 of the cases ($\approx 10\%$ of trials which finished). However, there is still room to improve, since in the remaining cases, `sb+s` uses up to $2.8\times$ the passes of the optimal solution.

The second set, `simple-tubes`, contains all 2113 problems on eight-loop tubes, where pairs of adjacent loops can either remain adjacent, be stacked atop each other, or be separated by an empty needle, and the overall tube can be rotated. These mimic the basic shaping operations used, e.g., by [10]; and, thus, the problems that `cse` was designed to solve. Figure 5 shows that `cse` produces optimal solutions in a much smaller fraction of problems: only 36 ($\approx 1.7\%$) were optimal, and solutions were sometimes more than $4\times$ slower. This is not unexpected, as `cse` may require up to $O(n^2)$ passes in the worst case [7].

For the final set, `cable-tubes`, we constructed 1183 transfer sequences by prepending 1x1 and 2x2 cables (loop crossings) to plans generated by `cse` for eight-loop tubes with rotations. The comparison with optimal transfer sequences (Figure 6) shows that there are significant fabrication speed gains to be made by combining cable and rotation transfers instead of performing them sequentially, as might be done by a programmer putting together an ad-hoc solution.

b) Canonical Node: We also examined the effect of canonicalizing on the time and memory requirements of the search on `all-short`, the subset of all datasets for which our search found a solution in $< 100ms$. As can be seen in Figure 7, the number of node expansions improves by a factor of $3165\times$, and runtime by $3828\times$, with larger problems experiencing more improvement. Of the few problems where canonical-node is slower (34 out of 5902 total problems), much of the slow down can be attributed to additional overhead from the canonicalize operation. We conjecture that

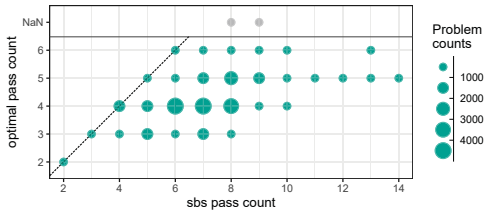

Fig. 4: The Schoolbus + Sliders (`sb+s`) algorithm [7] produces transfer plans within a factor of $3\times$ of optimal on a set of 6-loop lace-like patterns (`flat-lace`).

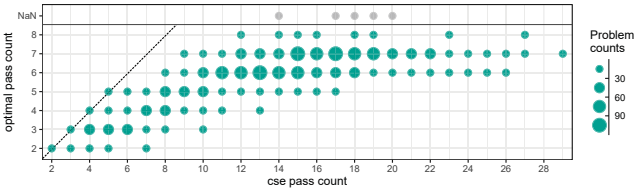

Fig. 5: The Collapse-Shift-Expand (`cse`) algorithm [6] produces transfer plans that stray relatively far from optimal on a set of 8-loop shaped tube problems (`simple-tubes`).

the 10 problems where canonical-node expands more nodes is due to tie breaking between equally weighted states.

c) Heuristics: In addition, we looked at the performance of various heuristics for the A* search (Table I). Our combined offset table and braid word length heuristic provide a three-order-of-magnitude reduction in both memory usage and search time compared to using no heuristic, and a one-order-of-magnitude reduction compared to the simpler combined braid word length and log offset heuristic. Building the offset table took 5.2 seconds, making it a strict improvement for larger problems even without amortizing the time required to build the table across multiple problems. Furthermore, taking the maximum of braid word length (which only looks at the braid) and offset table (which only looks at loop positions) provides improvements over using either heuristic alone.

VII. Discussion and Future Work

We presented a representation of a knitting machine's state that completely captures the effects of its transfer operations while being discrete and simple to update. We believe our work provides a crucial foundation for a host of interesting problems in the domain of knitting machine pattern generation.

While our A* search implementation is a breakthrough

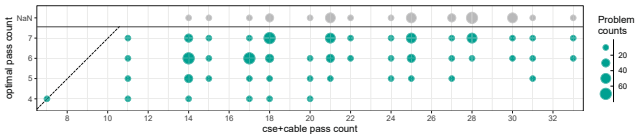

Fig. 6: The ad-hoc strategy of concatenating cable and tube rotation plans (dataset `cable-tubes`) presents many opportunities for algorithmic optimization.

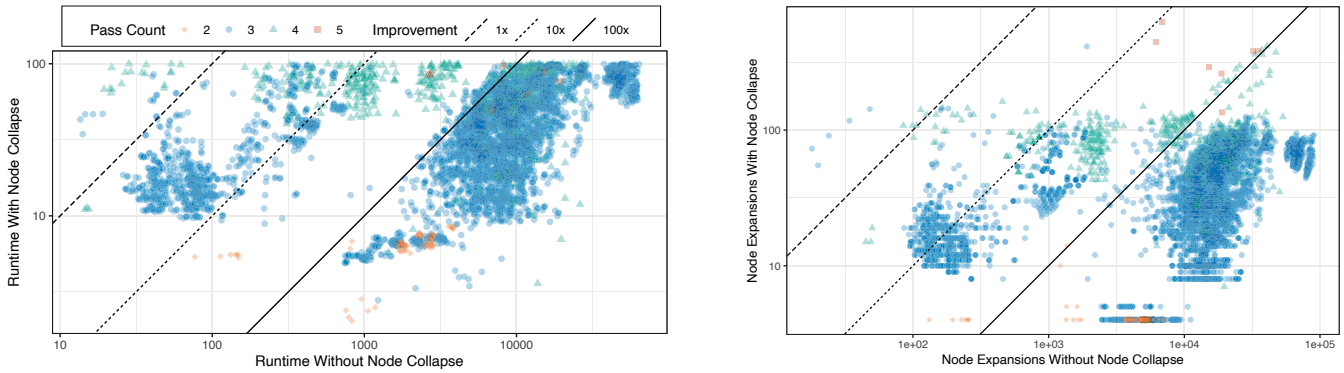

Fig. 7: Our canonical-node optimization results in an approximately two-order-of-magnitude reduction in both the search runtime and in the number of nodes expanded in our tests on the `all-short` dataset.

Scenario	Nodes	Time (s)	Speedup
No Heuristic	471451589	582264.3	1×
+ Braid	281291965	330931.9	1.8×
+ Log Offsets	2500142	2612.2	223×
+ Braid+Log	1663664	1688.6	344×
+ Prebuilt	259985	274.1	2124×
+ Braid+Prebuilt	184265	189.8	3067×

TABLE I: Total sum of nodes expanded and time taken for various combinations of heuristics over the 5002 problems that all heuristics finished. Heuristics were combined using $max(h_1, h_2)$.

in the optimality of existing planning algorithms, it still faces challenges in producing practical transfer plans. The exponential nature of the state expansion quickly makes A* search untenable for large problems. Memory issues are present even for problems with eight loops; meanwhile, knitting machines commonly use hundreds of loops at a given time. There exist several general approaches for dealing with high-dimensional search problems [19, 20]. It would be interesting to consider these approaches as well as methods that exploit the typically regular nature of knitting patterns; certain macro-level operations are quite common (e.g., rotate the tube two needles clockwise, cross every fifth loop over its neighbor to the right) while others are less so (e.g., close a tube by tangling loops around each other). A principled method to merge smaller subproblems could result in a practical transfer planning solution for typical large problems or provide tighter heuristics for optimal search.

Furthermore, in our work, we optimize solely the fabrication time of transfer plans. However, there exist other metrics a plan could account for, such as reliability. While knitting machines are generally quite robust, each transfer operation does have a small possibility of failure, with certain operations such as transferring large stacks of loops being more failure-prone, so minimizing the number of risky transfers could be another optimization goal. Other metrics that would improve the reliability of plans include minimizing the overall distance between loops, and considering the strain caused by twisting loops around each other.

In addition, while transfer planning answers the question of how to transition from an initial layout of loops on the machine to a new target layout, there is also the question of how to assign loops to machine needles in the first place. This loop assignment problem is also known as *scheduling*. While prior work [10] introduced an algorithm that guarantees a correct solution on certain subsets of the pattern space, the overall problem of correct, optimal scheduling is still open. It remains to be seen whether transfer planning and scheduling can be solved in concert to optimize overall pattern efficiency.

Another natural question that emerges is how to completely capture the result of all of a machine's operations, i.e., whether two general knitting patterns produce the same result. Aside from the transfer related operations presented here, there are *knit* and *tuck*, which create loops, *drop*, which unravels loops, and *miss*, which can potentially inlay yarn between loops. A representation that can fully account for all machine operations would pave the way for smart knit compilers capable of optimizing any knitting program.

VIII. Conclusion

Our braid-based representation is the first discrete representation that can fully capture the behavior of yarn during complex transfer operations. This opens the space of transfer planning to automated verification and search, providing important tools to simplify knit programming and optimization. Further, we think that our braid-based machine state representation provides a good starting point for a complete, canonical, discrete representation for *all* machine knitting operations. We hope that this work provides a representation amenable to the vast literature of planning techniques (of which we barely scratched the surface), effectively extending their reach into the realm of automated knitting transfer sequence design.

References

[1] M. Hofmann, L. Albaugh, T. Sethapakadi, J. Hodgins, S. E. Hudson, J. McCann, and J. Mankoff, "Knitpicking textures: Programming and modifying complex knitted textures for machine and hand knitting," in *Proceedings*

of the 32nd Annual ACM Symposium on User Interface Software and Technology, 2019, pp. 5–16.

[2] M. Popescu, L. Reiter, A. Liew, T. Van Mele, R. J. Flatt, and P. Block, "Building in concrete with an ultra-lightweight knitted stay-in-place formwork: prototype of a concrete shell bridge," in *Structures*, vol. 14. Elsevier, 2018, pp. 322–332.

[3] M. Popescu, M. Rippmann, A. Liew, L. Reiter, R. J. Flatt, T. Van Mele, and P. Block, "Structural design, digital fabrication and construction of the cable-net and knitted formwork of the knitcandela concrete shell," in *Structures*. Elsevier, 2020.

[4] J. Sabin, D. Pranger, C. Binkley, K. Strobel, and J. L. Liu, "Lumen," *Proceedings of the IASS Symposium*, 2018.

[5] Y. P. M. D. Moritz and M. A. Baranovskaya, "Knit-flatable architecture - pneumatically activated preprogrammed knitted textiles," in *eCAADe*, 2016.

[6] J. McCann, L. Albaugh, V. Narayanan, A. Grow, W. Matusik, J. Mankoff, and J. Hodgins, "A compiler for 3d machine knitting," *ACM Trans. Graph.*, vol. 35, no. 4, July 2016. [Online]. Available: https://doi.org/10.1145/2897824.2925940

[7] J. Lin, V. Narayanan, and J. McCann, "Efficient transfer planning for flat knitting," in *Proceedings of the 2nd ACM Symposium on Computational Fabrication*, ser. SCF '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3213512.3213515

[8] Shima Seiki, "Sds-one apex3," [Online]. Available from: http://www.shimaseiki.com/product/design/sdsone_apex/flat/, 2011.

[9] A. Kaspar, T.-H. Oh, L. Makatura, P. Kellnhofer, and W. Matusik, "Neural inverse knitting: From images to manufacturing instructions," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 3272–3281. [Online]. Available: http://proceedings.mlr.press/v97/kaspar19a.html

[10] V. Narayanan, L. Albaugh, J. Hodgins, S. Coros, and J. McCann, "Automatic machine knitting of 3d meshes," *ACM Trans. Graph.*, vol. 37, no. 3, pp. 35:1–35:15, Aug. 2018. [Online]. Available: http://doi.acm.org/10.1145/3186265

[11] A. Karmon, Y. Sterman, T. Shaked, E. Sheffer, and S. Nir, "Knitit: A computational tool for design, simulation, and fabrication of multiple structured knits," in *Proceedings of the 2nd ACM Symposium on Computational Fabrication*, ser. SCF '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3213512.3213516

[12] M. Hofmann, J. Mankoff, and S. E. Hudson, "Knitgist: A programming synthesis toolkit for generating functional machine-knitting textures," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1234–1247. [Online]. Available: https://doi.org/10.1145/3379337.3415590

[13] V. Narayanan, K. Wu, C. Yuksel, and J. McCann, "Visual knitting machine programming," *ACM Trans. Graph.*, vol. 38, no. 4, July 2019. [Online]. Available: https://doi.org/10.1145/3306346.3322995

[14] A. Kaspar, L. Makatura, and W. Matusik, "Knitting skeletons: A computer-aided design tool for shaping and patterning of knitted garments," in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 53–65. [Online]. Available: https://doi.org/10.1145/3332165.3347879

[15] P. Dehornoy, "Efficient solutions to the braid isotopy problem," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3091 – 3112, 2008, applications of Algebra to Cryptography. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X08000437

[16] D. B. A. Epstein, M. S. Paterson, J. W. Cannon, D. F. Holt, S. V. Levy, and W. P. Thurston, *Word Processing in Groups*. Natick, MA, USA: A. K. Peters, Ltd., 1992.

[17] K. Murasugi and B. Kurpita, *A Study of Braids*, ser. Mathematics and Its Applications. Springer Netherlands, 2012. [Online]. Available: https://books.google.com/books?id=VLTnCAAAQBAJ

[18] A. Felner, R. E. Korf, R. Meshulam, and R. C. Holte, "Compressed pattern databases," *Journal of Artificial Intelligence Research*, vol. 30, pp. 213–247, 2007.

[19] K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev, "Path planning with adaptive dimensionality," in *Fourth annual symposium on combinatorial search*, 2011.

[20] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.