D:Wave
The Quantum Computing Company™

Zephyr Topology of D-Wave Quantum Processors

TECHNICAL REPORT

Kelly Boothby, Andrew D. King, Jack Raymond

2021-09-22

Overview

This paper presents an overview of the topology of D-Wave's upcoming generation of quantum annealing processors. It provides examples of minor embeddings and discusses performance of embedding algorithms for the new topology compared to the existing Chimera and Pegasus topologies. It also presents some initial performance results for simple, standard Ising model classes of problems.

CONTACT

Corporate Headquarters
3033 Beta Ave
Burnaby, BC V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E Bayshore Rd
Palo Alto, CA 94303

Email: info@dwavesys.com

www.dwavesys.com

14-1056A-A
D-Wave Technical Report Series

Notice and Disclaimer

This document describes D-Wave’s plans at the time of publication regarding a future product. The design, features, functionality, and release date of the future product are subject to change. Nothing in this document is a representation, guarantee, or warranty of any aspect of the future product upon being made available by D-Wave or thereafter.

Contents

1 Introduction

This paper describes D-Wave's upcoming annealing processor topology; that is, the pattern that defines how the processor's qubits and couplers interconnect.

Zephyr is a significant advancement over D-Wave's previous *Pegasus* and *Chimera* topologies, available in the Advantage and D-Wave 2000Q products respectively. Zephyr features qubits of degree 20 and native K_4 and $K_{8,8}$ subgraphs.

2 The Zephyr Topology

In Zephyr, as in Pegasus and Chimera, qubits are *oriented* vertically or horizontally. Chimera has two types of coupler: internal couplers connect pairs of orthogonal (with opposite orientation) qubits, and external couplers connect colinear pairs of qubits (that is, pairs of qubits that are parallel, in the same row or column). The Pegasus family has, in addition to Chimera's internal and external couplers, a third type: odd couplers. Odd couplers connect parallel qubit pairs in adjacent rows or columns. The Zephyr topology features these three coupler types, with a total of two odd couplers, two external couplers, and sixteen internal couplers.

In the Chimera topology, qubits have a *nominal length* of 4 (each qubit is connected to 4 orthogonal qubits through internal couplers) and degree of 6 (each qubit is connected to 6 different qubits through couplers). In the Pegasus family, qubits have a nominal length of 12 and degree of 15. In the Zephyr topology, qubits have a nominal length of 16 and degree of 20.

2.1 Formulaic Description

In broad strokes, a Zephyr graph with *grid parameter* m and *tile parameter* t, notated $Z_{m,t}$ (or simply Z_m when $t = 4$), contains $8tm^2+4tm$ qubits, and has maximum degree $4t+4$. For example, a Z_{15} graph contains 7440 qubits in total. Abstractly speaking, a Zephyr graph can be constructed by first constructing a Chimera graph $C_{2m+1,2m+1,t}$, adding odd couplers, joining qubits into pairs along tile boundaries in an alternating fashion, and finally deleting unpaired qubits on the periphery. This construction is shown in 1, but not elaborated in full rigor.

We enumerate the qubits of Z_m with vectors of length 5, where a qubit (u, w, k, j, z) has coordinates

- u is the *orientation*, indicating if a qubit is vertical ($u = 0$) or horizontal ($u = 1$).
- w is the *perpendicular block offset*, indicating the index of the qubit's tile, in the orientation perpendicular to u, with $0 \leq w < 2m + 1$. (That is, if $u = 0$, then w is a horizontal (column) index, and if $u = 1$, then w is a vertical (row) index.)
- k is the *qubit index*, indicating the index of a qubit within a tile, and $0 \leq k < t$.

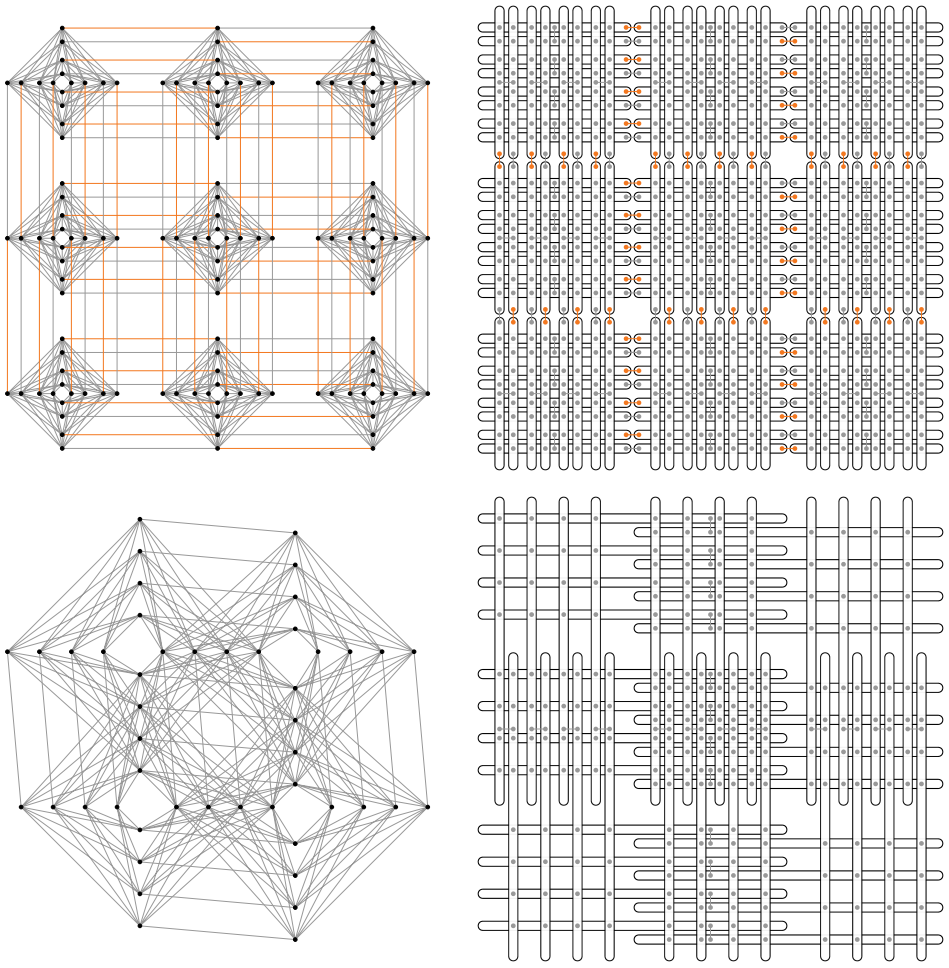

Figure 1: Construction of the Zephyr lattice Z_1 from a Chimera $C_{3,3,8}$. **(top left)** The $C_{3,3,8}$ graph in the point-and-line drawing style, with black nodes, edges to be contracted in orange, and remaining edges grey. **(top right)** The same $C_{3,3,8}$ graph drawn in qubit-loop style, with internal edges drawn as grey circles, and odd and external couplers drawn as barbells colored as above. **(bottom left)** The Z_1 graph drawn in point-and-line style. **(bottom right)** The Z_1 graph drawn in qubit-loop style.

- j is the *shift identifier*, with $j = 0$ to indicate that a qubit is shifted to the left/top and $j = 1$ to indicate that a qubit is shifted to the right/bottom.
- z is the *parallel tile offset*, indicating the index of the qubit's tile in the orientation parallel to u, with $0 \leq z < m$. (That is, if $u = 0$, then z is a vertical (row) index, and if $u = 1$, then z is a horizontal (column) index.)

In the following description of the three types of couplers, a coupler $p \sim q$ exists whenever both p and q are qubits in $Z_{m,t}$. The three sets of Zephyr couplers are:

- *external*: $(u, w, k, j, z) \sim (u, w, k, j, z + 1)$
- *odd*: $(u, w, k, 0, z) \sim (u, w, k, 1, z - \alpha)$ for $\alpha \in \{0, 1\}$
- *internal*: $(0, 2w + 1 - \alpha, k, j, z - j\beta) \sim (1, 2z + 1 - \beta, h, i, w - i\alpha)$ for $\alpha \in \{0, 1\}$ and $\beta \in \{0, 1\}$

We define an integer labeling for Zephyr via the function

$$(u, w, k, j, z) \mapsto z + m(j + 2(k + t(w + (2m + 1)u))),$$

which is a bijection between the $8tm^2 + 4tm$ coordinate labels and the interval

$$\{0, \cdots, 8tm^2 + 4tm\}.$$

2.2 Clique and Bipartite Embeddings for Problem Solving

As we did with the Pegasus family, we use the fact that Zephyr (with odd couplers removed) can be viewed as a graph minor of Chimera to ease the computation of efficient clique embeddings. Optimal embeddings of complete graphs, $\varepsilon : K_a \to Z$ consist of a parallel paths of horizontal qubits and a parallel paths of vertical qubits, connected at a point of intersection to make a chains. The algorithm of [1] can be modified to produce embeddings with chains of length m. For graphs with full yield, the resulting embedding is equivalent (albeit with shorter chains) to an embedding in a full-yield Chimera $C_{2m-1,2m-1,2t}$ (the square Chimera graph obtained by deleting the peripheral cells of $C_{2m+1,2m+1,2t}$). Thus, the size of this embedding is $2(2m - 1)t$, or $16m - 8$ when t is 4.

Likewise, the largest useful complete bipartite graphs are obtained by working within the non-peripheral square. Complete bipartite embeddings $\varepsilon : K_{a,b} \to Z$ consist of a parallel paths of horizontal qubits, each in a different row, and b parallel paths of vertical qubits, each in a different column. This gives an embedding of $K_{2(2m-1)t,2(2m-1)t}$ (or $K_{16m-8,16m-8}$ when t is 4), where all chains have length m.

3 Lattice Embeddings

Recent work has shown great promise in the simulation of two- and three-dimensional quantum magnetic systems using quantum annealers [2, 3]. In Pegasus, embedding of 3D lattices

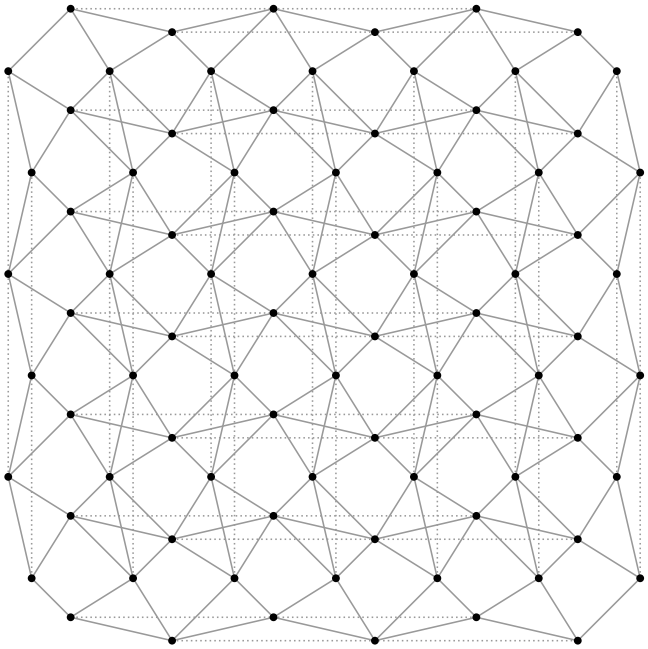

Figure 2: Example of a Zephyr graph $Z_{3,1}$, with external edges drawn as dotted lines. This graph, omitting the external edges, is a checkerboard lattice, resulting in a variety of possible direct lattice embeddings, with no chains required.

was simplified from four-qubit chains to two-qubit chains, and leading to a scaling improvement in solution time [4]. Similarly, 3D lattices can be embedded in Zephyr using two-qubit chains. However, several examples show that embedding low-dimensional lattices in Zephyr is easier than in Pegasus. Fig. 2 shows a $Z_{3,1}$ graph, wherein a direct embedding of a checkerboard lattice is clearly visible. In Chimera and Pegasus, qubit spin ice [5] has been implemented using four-qubit chains and two-qubit chains, respectively. In Zephyr, this can be done with one-qubit chains, with no translation between the logical and embedded systems. As an immediate consequence of the direct embedding of a checkerboard lattice, we can also directly embed the square lattice and the Shastry-Sutherland lattice [6]. Furthermore we can directly embed in Zephyr the lattice gauge theory described in [7], whereas embedding in Chimera took four-qubit chains [8]. Thus we can expect the Zephyr topology to provide a rich testbed for effectively probing quantum effects in materials.

4 Heuristic Embedding

We compare the Zephyr topology with the Pegasus and Chimera topologies by examining the results of heuristic embedding of problems into each of the three topologies. In an effort to make fair comparisons, we consider two topologies roughly equivalent when they have the same *width*: the number of horizontal (equivalently, vertical) nodes in the graph that results from contracting external couplers. The formulas for widths are $16m + 8, 12m - 4$, and $4m$ for Z_m, P_m, and C_m respectively (note, the peripheral components of Pegasus P_m are removed). The heuristic embedding algorithm used for this study is `minorminer` version 0.2.5, denoted $\mathbf{A}$ below.

4.1 Methodology

For problem s and topology T, we write $c \in \mathbf{A}(s, T)$ for chain c produced by algorithm $\mathbf{A}$ for embedding $s \to T$. For a fixed number of trials, t, we define three metrics,

- *average chain length*: $\ell(s, T) = \frac{1}{t} \sum_{i=1}^{t} \frac{1}{|s|} \sum_{c \in \mathbf{A}(s,T)} |c|$,
- *maximum chain length*: $L(s, T) = \frac{1}{t} \sum_{i=1}^{t} \max_{c \in \mathbf{A}(s,T)} |c|$,
- *average runtime*: $\tau(s, T)$ is the time taken to produce the t embeddings, $\mathbf{A}(s, T)$, divided by t.

Note that heuristic embedding algorithms cannot generally be expected to produce embeddings every time. To make these metrics sensible, we execute $\mathbf{A}(s, T)$ until we have accumulated $t = 100$ embeddings and record the total time spent including failures. If we fail to produce 100 embeddings for any graph in a family, into any target topology, we discard the entire family from our results.

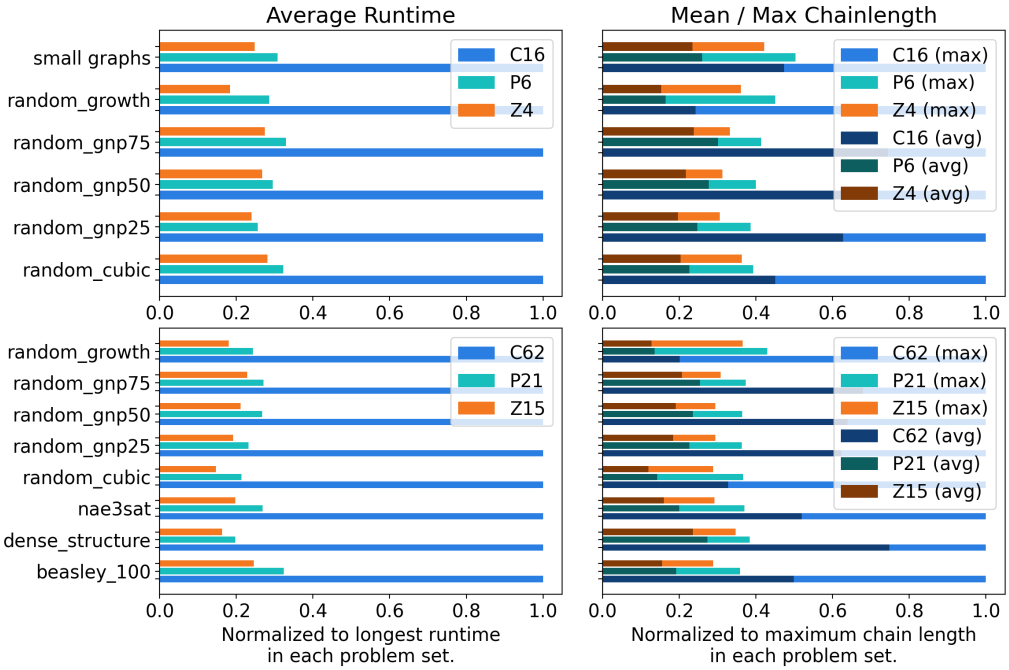

Figure 3: Heuristic embedding benchmark results. See Section 4 for more details.

4.2 Results

Figure 3 shows summary data for comparison of small and large topologies. The small topologies have the same width as the D-Wave 2000Q C_{16} (compared against Z_4 and P_6), and the large topologies have the same width as a projected Z_{15} product (compared against P_{21} and C_{62}). In both cases, we have normalized the scores in each graph family to the greatest measure across the three topologies.

The following problem sets were chosen for their diversity, to ensure that these trends are not a result of properties of a particular problem set:

- small graphs: a collection of named graphs found in computer algebra systems such as Sage and Mathematica; the largest being the 126-node Tutte 12-cage, path and star graphs with up to 100 nodes, complete bipartite graphs with up to 20 nodes, grid graphs with up to 100 nodes, and hypercubes with dimension up to 7; labeled `small_graphs`.
- Beasley 100: graphs from the Beasley [9] max-cut dataset with 100 nodes, labeled `beasley_100`.
- dense structured graphs: complete graphs K_n, complete bipartite graphs $K_{n,n}$ and circular complete graphs $K_{4n/4}$, labeled `dense_structure`.
- not-all-equal-3SAT graphs near the critical threshold, labeled `nae3sat`.

- Erdös-Rényi random graphs, $G(n, p)$,
  - $G(70, .25)$ (labeled `random_gnp25`),
  - $G(60, .50)$ (labeled `random_gnp50`), and
  - $G(50, .75)$ (labeled `random_gnp75`).
- random cubic graphs: graphs with uniform degree 3, labeled `random_cubic`.
- random growth: graphs generated through the `duplication_divergence_graph` generator from `networkx` with $p = 1/3$, labeled `random_growth`.

Overall, we see a marginal improvement in Zephyr over Pegasus, around 2-10% improvement in both runtime and chain lengths.

5 Treewidth

One measure of the complexity of a graph is its treewidth [10]. For example, the minimum energy of an Ising model defined on a graph of treewidth τ with n vertices can be found in time $O(n2^\tau)$ using dynamic programming [11]. Here we show that the treewidth of the Zephyr Z_m graph is between $16m$ and $16m+8$. For comparison, the treewidth of a Chimera C_m graph is $4m$. In both cases, the treewidth is roughly the number of rows (or columns) of qubits as described in Section 5.1.

To lower-bound the treewidth, consider embeddable complete graphs. A complete graph K_n has treewidth $n-1$, and we will show that a complete graph embedding of size $16m+1$ can be embedded in Z_m in Section 5.1. It is possible to upper-bound the treewidth by $16m+8$ by providing a *vertex elimination order* (see [12, Theorem 6] for background). One such vertex elimination order is as follows:

- eliminate vertical qubits, one parallel path column at a time;
- eliminate horizontal qubits in any column once all vertical qubits adjacent to them have been eliminated.

5.1 Clique Embedding for Treewidth Estimate

In this section, we briefly show a family of clique embeddings which demonstrate that cliques of size $16m+1$ can be embedded into Z_m with $m > 1$. Almost all chains in these embeddings contain $2m$ qubits, and are expected to perform significantly worse than the embeddings described in Section 2.2, which have shorter chain lengths. The purpose of this exposition is to support a computation of the *treewidth* complexity measure.

We construct our family of embeddings with a set of *chain descriptors*. Each chain descriptor is a set of 4-tuples (u, w, k, j) which corresponds to the *track* $c(u, w, k, j) = \{(u, w, k, j, z) : 0 \leq z < m\}$. For $m > 1$, let

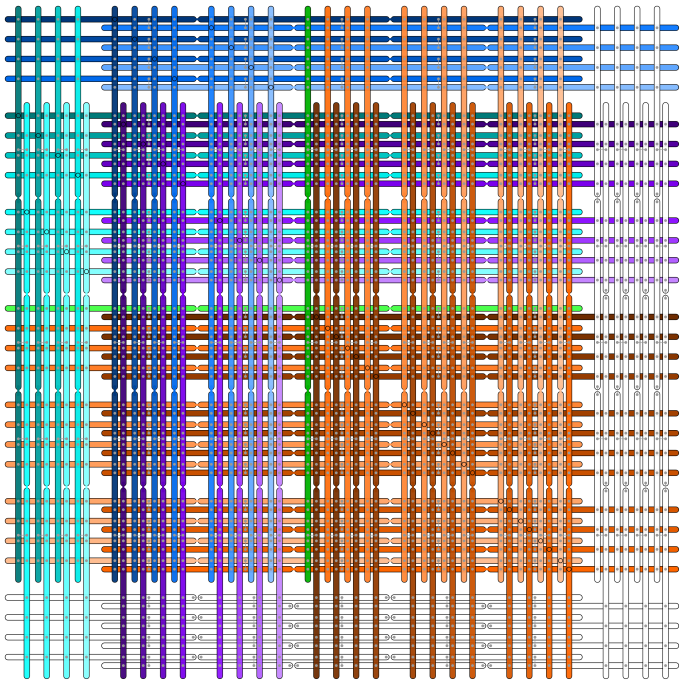

Figure 4: Example of a K_{49} embedded into a Zephyr graph Z_3, as shown in Section 5.1, in the qubit-loop style. Odd and external edges within a chain are drawn thicker in the same color as the chain; internal edges within a chain are drawn with a larger black circle. Chains are colored according to the set they belong to; A: , B: , C: , D: , E: .

$$A = \{\{(0,3,0,0)\},\{(1,3,0,0)\}\},$$
$$B = \{\{(0,w,k,1),(1,w,k,1)\} : 1 \le w < 3, 0 \le k < 4\},$$
$$C = \{\{(0,0,k,j),(1,1+j,k,0)\} : 0 \le k < 4, 0 \le j < 2\},$$
$$D = \{\{(0,1+j,k,0),(1,0,k,j)\} : 0 \le k < 4, 0 \le j < 2\},$$

and

$$E = \{\{(0,w,k,j),(1,w,k,j)\} : 3 \le w < 2m, 0 \le k < 4, 0 \le j < 2 \text{ and } w+k+j > 3\}.$$

Expanding the chain descriptors of A, B, C, D, and E (with 2, 8, 8, 8, and $16m-25$ descriptors respectively) into chains, we get an embedding of K_{16m+1} into Z_m. See Figure 4 for this embedding in Z_3. Note that this embedding is chosen in part due to the simplicity of its description. It is possible to modify the embedding to produce two chains of length m and $16m-1$ chains of length $m+2$, by modifying E to join tracks mostly along an antidiagonal and judiciously trimming all chains not in A.

6 Binary Quadratic Models

In this section we examine some basic static properties and heuristic performance insights for binary quadratic model optimization on the Zephyr topology, comparing to other processor architectures. In Section 6.1 we present a detailed analysis of the Hamze-deFreitas-Selby (HFS) algorithm [13–15]. This is a powerful heuristic for Chimera lattice problems, but relatively impractical for Pegasus and Zephyr.

For a graph defined by vertex and edge sets, $\{V, E\}$, a classical Ising model over N spins $s = \{-1, 1\}^N$ is defined by a Hamiltonian $H = \sum_{ij \in E} J_{ij} s_i s_j + \sum_{i \in V} h_i s_i$; this is a special (but sufficiently general) case of a binary quadratic model that is the focus of this section. In a standard quantum annealing protocol, the problem Hamiltonian ($\hat{H}_P$) is encoded with each spin s_i replaced by a Pauli operator σ_i^z, and acts alongside quantum fluctuations induced via a transverse field driver Hamiltonian $\hat{H}_D = -\sum_i \sigma_i^x$. The operator in effect during the anneal is $\hat{H} = \mathscr{J}\hat{H}_P + \Gamma\hat{H}_D$. When a quantum processor is used as an annealer we begin with a large driver term Γ (and small problem term, $\mathscr{J}$). We then smoothly decrease (increase) these terms so that the wavefunction concentrates over low-energy states of the problem Hamiltonian, where it is measured to provide an estimate to the ground state.

Practical competitors to a quantum processor consist primarily of classical heuristics based upon Markov chain methods [16–18]. Some heuristics can be tailored to exploit lattice structure, other heuristics are topology agnostic. The largest advantage from annealing is anticipated in problems with the largest number of degrees of freedom; for this reason we evaluate problems defined directly over the processor architecture, alongside lattice-leveraging heuristic competitors. To understand performance we take the simplest possible exemplar for hard problems over Zephyr, Pegasus and Chimera topologies: the standard spin-glass problem with independent and identical distributed couplings $J_{ij} = \pm 1\ \forall ij \in E$ and $h_i = 0\ \forall i \in V$. This problem is anticipated to have a zero temperature (and finite transverse field) spin-glass phase transition, making determination of a ground state challenging [19].

The open-source Ocean SDK package provides a simulated thermal annealing (SA) implementation, `dwave-neal` [20]. For SA implemented with `dwave-neal` we can use the default geometric schedule $T = T_{max}(T_{min}/T_{max})^{b/(B-1)}$ with $b = 0 \dots B-1$. $T_{max} = k/\log(2)$ for maximum lattice connectivity k (ensuring fast mixing in the initial algorithm stage) and $T_{min} = 2\log(100N)$ (ensuring excitations are absent in returned samples with high probability). For the case of SA an important figure of merit is the time per sweep: the time required to attempt a single Metropolis update on every variable. We use this value to infer some performance limitations in SA, and also on closely related heuristics such as parallel tempering, population annealing, and path-integral Monte Carlo.

Ocean SDK also provides a steepest greedy descent (SGD) implementation, `dwave-greedy`, a useful heuristic for optimization and inference. Typically, applications use a very large number of reads, where wallclock time is proportional to number of samples, so the time per sample is a useful figure of merit.

We evaluate the figures of merit at full processor scale, defect-free over the largest graph component in Table 1. The maximum scales for D-Wave 2000Q and Advantage processors are C_{16} and P_{16} respectively, whereas a future Zephyr-based processor is projected to be Z_{15}. For `dwave-neal` we set $B = 1024$ and allow 256 sweeps at each beta and consider time for a single sample. For `dwave-greedy` we draw 1024 samples. We present results averaged over 25 model instances in the table. Times are measured single threaded on a Intel® Core™i7-8665U CPU.

Lattice	`dwave-neal` time (s) per sweep	`dwave-greedy` time (s) per sample
Chimera C_{16}	4.0×10^{-5}	1.1×10^{-3}
Pegasus P_{16}	1.4×10^{-4}	1.2×10^{-2}
Zephyr Z_{15}	2.3×10^{-4}	1.7×10^{-2}

Table 1: Simulated thermal annealing and steepest greedy descent operation time scales

Due primarily to increased size, and in part owing to increased connectivity, time scales are longer for the proposed Zephyr architecture. The SA and SGD algorithms are amenable to lattice-orientated parallelization [17]. Given that GPU implementations can be communication and memory limited, we anticipate this computational platform to be relatively less effective in supporting the Zephyr architecture where spatial division of variable sets creates relatively densely connected blocks with high external connectivity. Beyond these intuitive observations, lattice-specific platform choice and optimization is beyond the scope of this section.

In Section 6.1 we evaluate an HFS method, which is significantly different from `dwave-neal` and `dwave-greedy`, and is competitive as a heuristic optimizer in the case of Chimera-structured binary quadratic models. This method is not effective for Zephyr. Very generally, lattice-leveraging heuristics are more costly when applied to the more highly connected Zephyr architecture. An example is the recently proposed tensor-network based method for lattice inference [21]. This heuristic exploits the fact that one can process a Chimera lattice cell-row by cell-row, where cells are $K_{4,4}$ in type, efficiently compressing and passing information in the form of (cell-level) matrix product states. For Zephyr the higher connectivity

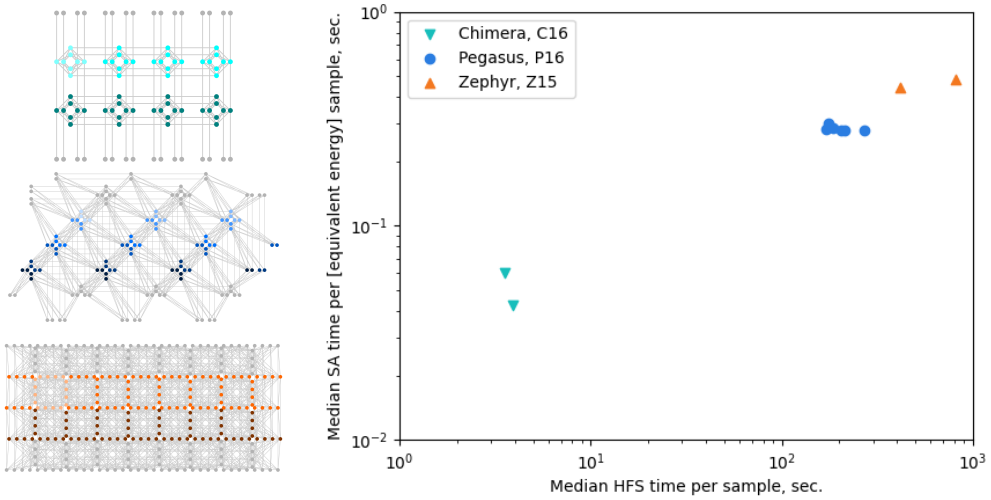

Figure 5: (left) HFS algorithms propose energy minimizing moves of large row (or column) like regions. By applying this process iteratively we potentially optimize the full problem. As the (geometric) thickness of rows increases, the cost of operating the move grows exponentially, so the algorithm is limited to narrow rows. Higher connectivity within rows also raises the cost, and connectivity outward from the row weakens effectiveness, hence moves in Zephyr are both more costly and less effective than in previous processor generations. Examples from top to bottom are shown for Chimera (C4), Pegasus (P4), and Zephyr (Z4), where shading indicates the rows (and blocks) underlying a region. **(right)** We evaluate the effectiveness of an HFS scheme relative to SA (time to achieve equal energy). The HFS implementation studied here is much slower than SA. Furthermore, in moving from Chimera to Pegasus and Zephyr, the cost of HFS methods increases by a large amount, but achieving equal quality outcomes requires only marginally longer in SA. It has been demonstrated that optimization in Chimera can substantially reduce this time, but optimization for the more complicated moves in Pegasus and Zephyr is challenging.

implies that a similar construction would naively need cells of treewidth 17 as opposed to 4, with a corresponding exponential penalty in the use of tensor-networks. Without considering the practical difficulties of efficient implementation, this theoretical slowdown already makes the method impractical. In the architecture change from Chimera to Pegasus [22], we studied several static properties and heuristic algorithms. Results presented therein are also expected to extend qualitatively to Zephyr, as a function of increased connectivity. As one example, parallel tempering with inter cluster moves (ICM) were examined [23]. These are less effective in Pegasus than Chimera, a conclusion we expect to hold for Zephyr, since impactful ICMs are more efficiently constructed at lower connectivity.

6.1 The HFS Algorithm

Standard implementations of SGD, SA and related algorithms search a space by considering energy changes as a function of single variable moves. The power of HFS methods arises from considering moves over large variable sets [13–15], sets which (on average) have strong

internal correlations and weaker external correlations. In most topologies, updates come at an exponential cost in the set size, or it is simply impossible to isolate large sets with strong internal (relative to external) coupling, offsetting any benefit. However, in the context of processor architectures the regular cell-like structure allows for an intuitive identification of such regions, and in context of Chimera, algorithms based on these set choices have proved to be powerful. In consideration of the last architecture change from Chimera to Pegasus [22], we analyzed the scaling for several heuristics. This included an analysis of HFS methods where regions were constructed on-the-fly by a heuristic. However, to make best use of HFS it is valuable to exploit knowledge of the regular static lattice structure as we do in the analysis of this section.

We construct small, strongly connected sets of variables (regions, R) for processor architectures as shown in Figure 5 — by correspondence with methods that have proved successful over Chimera [14]. For Chimera we construct an elemental row by joining $K_{4,4}$ blocks horizontally. For Pegasus we can construct an elemental row by joining K_4 blocks horizontally. For Zephyr we join $K^+_{8,8}$ blocks horizontally (where $K^+_{8,8}$ denotes a $K_{8,8}$ subgraph along with four odd couplers per orientation). Although thinner rows are possible, subdivision of the cells performs poorly since we want to keep strongly connected variables together (on average, they are strongly correlated), and there is no natural way to subdivide the blocks. From elemental row regions we can also create thicker regions by merging adjacent rows. The number of rows merged is the thickness. Moves over thicker rows are more powerful, but also exponentially more costly in the thickness. For an HFS scheme of given thickness we include the corresponding rows, along with columns of the same thickness, related by symmetry. We have a hierarchy of HFS schemes as a function of the thickness (thickness 1 or 2 for Chimera and Zephyr, or thickness up to 6 for Pegasus, see Figure 5). Thus we have 10 different schemes ($\mathscr{R}$) as a function of the lattice type and thickness, with each scheme defined by a set of regions.

Given an HFS scheme we evaluate a simple optimizer by iterating over the corresponding regions (Algorithm 1). We initialize our state uniformly at random. We then select a region (column or row) uniformly at random, and minimize the assignment over this region conditioned on the variable assignments outside the region. For a given scheme each spin is updated on average once every $S(\mathscr{R})$ updates, which can be considered a sweep. After performing a sweep with no decrease in energy we terminate the algorithm (updates are no longer efficiently yielding improved energies). This algorithm is didactic and captures qualitatively the relative power of different schemes.

The optimization step uses the bucket-tree elimination algorithm [11] implementation in C++ for generic graphs [24]. This algorithm requires an elimination order. For row regions we can make use of the standard elimination orders [20] restricted to the corresponding variables. For column regions we use the same orders (up to horizontal to vertical qubit symmetry transformation). These elimination orders are optimal[1], in the sense that they saturate the treewidth bound for the region, which can be brute forced independently for these examples.

Theoretical features of the HFS algorithms studied are enumerated in Table 2. Methods built from column and row regions of treewidth 4 and 8 have proven competitive against

[1] Except for boundary regions in Pegasus and Zephyr, where some inconsequential savings are possible with alternative orderings

Algorithm 1 Minimize N variable lattice problem: $H(x) = \sum_i h_i x_i + \sum_{ij} J_{ij} x_i x_j$

```
procedure Greedy-HFS(h, J, ℛ)
    Select uniform random x ∈ {−1, 1}^N
    num_unsuccessful_updates = 0
    while num_unsuccessful_updates < S(R) do
        select uniformly at random a region R from the set ℛ
        H_R(y) = Σ_{ij∈R} J_ij y_i y_j + Σ_{i∈R} [h_i + Σ_{j∉R} J_ij x_j] y_i
        x_R ← argmin(H(x_R))
        if update lowers energy then
            num_unsuccessful_updates = 0
        else
            num_unsuccessful_updates = num_unsuccessful_updates + 1
    return x
```

QPUs exploiting Chimera topology. Regions built for Pegasus and Zephyr are necessarily of higher treewidth. Optimization for a region of treewidth t requires memory scaling as 2^{t+1}. The successor architectures to Chimera necessarily involve slower updates.

QPU architecture	Row/Column Thickness	Treewidth
Chimera	[1,2]	[4,8]
Pegasus	[1,2,3,4,5,6]	[4,5,7,9,11,13]
Zephyr	[1,2]	[9,17]

Table 2: Algorithmic complexity properties of some HFS schemes

As a simple measure of utility for these schemes consider the energies returned per sample, versus the energy returned by SA. For a set of 25 problems on each processor architecture we draw one sample by the HFS heuristic providing, most likely, a local minima, with an associated wallclock time. We then run SA for the same problem, doubling the number of sweeps for a single sample until we match (or improve upon) the energy obtained by the HFS algorithm. We use the same geometric scheme and temperature bounds as described in the previous section for `dwave-neal`, doubling B at each stage with 1 sweep per temperature in the schedule. Both algorithms are implemented single threaded on a Intel® Core™i7-8665U CPU. We plot the time required by HFS against the time required by SA in Figure 5(right). We observe that these methods are both far more costly in the Zephyr and Pegasus processors and far less effective, relative to Chimera.

In this implementation, none of the methods (including those for Chimera) are competitive against SA in reaching low energies. More efficient implementations of HFS algorithms for Chimera graphs exist, such as that by Selby [13], and can outperform SA. These optimizations involve differences in coding, region sets and termination criteria, comparable methods might be attempted for Pegasus and Zephyr. As an example of a region difference, Alex Selby's HFS implementation for Chimera arises in combining rows and column regions through additional thin bridge regions. Such bridges do not qualitatively impact our conclusion, and we note they are relatively difficult to construct in Pegasus and Zephyr owing to more complex inter-cell connectivity. Furthermore, the C++ implementation is better

suited to large treewidth cases, so we anticipate less leeway to optimize for Zephyr than Chimera. We anticipate the two orders of magnitude gap in performance between Chimera and Zephyr would be larger in carefully optimized code, so that Zephyr based HFS methods would remain uncompetitive with SA despite the competitiveness of well optimized Chimera codes.

We have shown in our analysis that HFS is relatively inefficient in Zephyr and Pegasus. However, this is a concise didactic study, and it is important to note some caveats. (1) We use a specific generic graph software implementation for HFS, and a specific generic graph SA implementation, for wallclock timing. (2) The Chimera, Pegasus and Zephyr lattices compared in Figure 5(right) differ in size. We considered the cases with equally many qubits and the performance gap remained large. (3) We consider only one canonical random model per lattice type. Other models with particular structure may behave quite differently. (4) For Pegasus and Zephyr we also considered another natural set of regions, those stretched along diagonals as opposed to rows columns, and observed similar outcomes.

HFS methods are a rich family of techniques that can be used as part of sampling and optimization heuristics for certain lattices. We studied the performance of a hierarchy of intuitive HFS methods, qualitatively matching methods that proved successful over Chimera lattice problems. We used performance of SA as a comparison point. In Zephyr HFS methods are ineffective in absolute terms relative to SA, and relative to Chimera HFS methods. We anticipate that problems over a Zephyr topology will be less susceptible to optimized HFS implementations than comparable problems defined over Chimera and Pegasus lattices.

7 Conclusion

The architecture of D-Wave's upcoming generation of annealing processors introduces qubits with a higher degree of connectivity in the new Zephyr topology. In this paper we describe, and provide a formulaic description for, the new topology. We describe some key advantages over previous offerings, including:

- Cliques and bicliques. Zephyr Z_m supports embedding cliques of up to size $16m - 8$, with chains of length m, and bicliques up to $K_{16m-8,16m-8}$, with uniform chain length of m.
- Lattices. Zephyr supports 2D and 3D lattice embeddings with equal or shorter chain lengths than previous topologies.
- Heuristic embedding. Zephyr sees a marginal improvement in heuristic embedding performance when compared to Pegasus, and both topologies are a significant improvement over Chimera.
- Treewidth. Zephyr Z_m has a treewidth of between $16m$ and $16m + 8$; for comparison, treewidth of a Chimera C_m graph is $4m$.
- Binary quadratic models. Heuristic optimization and inference on Zephyr topology problems is difficult to accelerate by decomposition heuristics such as the HFS method, reflecting the greater complexity of patterns amongst the native spin degrees of freedom.

References

[1] K. Boothby, A. D. King, and A. Roy, "Fast clique minor generation in Chimera qubit connectivity graphs," Quantum Information Processing **15**, 495–508 (2016).

[2] R. Harris, Y. Sato, A. J. Berkley, M Reis, F. Altomare, et al., "Phase transitions in a programmable spin glass simulator," Science **165**, 162–165 (2018).

[3] A. D. King, J. Raymond, T. Lanting, S. V. Isakov, M. Mohseni, et al., "Scaling advantage over path-integral Monte Carlo in quantum simulation of geometrically frustrated magnets," Nature Communications **12**, 1113 (2021).

[4] A. D. King and W. Bernoudy, "Performance benefits of increased qubit connectivity in quantum annealing 3-dimensional spin glasses," (2020), arXiv:2009.12479.

[5] A. D. King, C. Nisoli, E. D. Dahl, G. Poulin-Lamarre, and A. Lopez-Bezanilla, "Qubit spin ice," Science, 10.1126/science.abe2824 (2021).

[6] P. Kairys, A. D. King, I. Ozfidan, K. Boothby, J. Raymond, A. Banerjee, and T. S. Humble, "Simulating the Shastry-Sutherland Ising Model Using Quantum Annealing," PRX Quantum **1**, 020320 (2020).

[7] C. Chamon, D. Green, and Z.-C. Yang, "Constructing Quantum Spin Liquids Using Combinatorial Gauge Symmetry," Physical Review Letters **125**, 067203 (2020).

[8] S. Zhou, D. Green, E. D. Dahl, and C. Chamon, "Experimental Realization of Spin Liquids in a Programmable Quantum Device," (2020), arXiv:2009.07853.

[9] J. E. Beasley, "Or-library: distributing test problems by electronic mail," Journal of the Operational Research Society **41**, 1069–1072 (1990), eprint: https://doi.org/10.1057/jors.1990.166.

[10] N. Robertson and P. Seymour, "Graph minors. II. Algorithmic aspects of tree-width," Journal of Algorithms **7**, 309–322 (1986).

[11] R. Dechter, "Bucket elimination: A unifying framework for reasoning," Artificial Intelligence **113**, 41–85 (1999).

[12] H. L. Bodlaender and A. M. Koster, "Treewidth computations i. upper bounds," Information and Computation **208**, 259–275 (2010).

[13] *QUBO-Chimera*, https://github.com/alex1770/QUBO-chimera, Accessed: 2021-07-19.

[14] A. Selby, "Efficient subgraph-based sampling of Ising-type models with frustration," (2014), arXiv:1409.3934.

[15] F. Hamze and N. de Freitas, "From fields to trees: On blocked and collapsed mcmc algorithms for undirected probabilistic graphical models," in Proc. uncertainty in artificial intelligence (2004).

[16] H. Oshiyama and M. Ohzeki, "Benchmark of quantum-inspired heuristic solvers for quadratic unconstrained binary optimization," 1–9, arXiv:arXiv:2104.14096v1.

[17] J. King, S. Yarkoni, J. Raymond, I. Ozfidan, A. D. King, M. M. Nevisi, J. P. Hilton, and C. C. McGeoch, "Quantum Annealing amid Local Ruggedness and Global Frustration," Journal of the Physical Society of Japan **88**, 61007 (2019), arXiv:1701.04579.

[18] C. Coffrin, H. Nagarajan, and R. Bent, "Evaluating Ising Processing Units with Integer Programming," in Integration of constraint programming, artificial intelligence, and operations research, edited by L.-M. Rousseau and K. Stergiou (2019), pp. 163–181.

[19] H. Nishimori, *Statistical physics of spin glasses and information processing: an introduction* (2001).

[20] *D-Wave Ocean software documentation*, https://docs.ocean.dwavesys.com/en/stable/, Accessed: 2021-07-19.

[21] M. M. Rams, M. Mohseni, and B. Gardas, "Heuristic optimization and sampling with tensor networks," arXiv, 1–13 (2018), arXiv:1811.06518.

[22] K. Boothby, P. Bunyk, J. Raymond, and A. Roy, "Next-Generation Topology of D-Wave Quantum Processors," (2020), arXiv:2003.00133.

[23] Z. Zhu, A. J. Ochoa, and H. G. Katzgraber, "Efficient Cluster Algorithm for Spin Glasses in Any Space Dimension," Physical Review Letters **115**, 1–5 (2015), arXiv:1501.05630.

[24] The bucket tree elimination routine used is pending publication as part of D-Wave Ocean SDK.