# Pipelined correlated minimum weight perfect matching of the surface code

Alexandru Paler[1,2] and Austin G. Fowler[3]

[1]Aalto University, Espoo 02150, Finland
[2]University of Texas at Dallas, Richardson, TX 75080, USA
[3]Google Inc., Santa Barbara, 93117 CA, USA

arXiv:2205.09828v2 [quant-ph] 11 Dec 2023

**We describe a pipeline approach to decoding the surface code using minimum weight perfect matching, including taking into account correlations between detection events. An independent no-communication parallelizable processing stage reweights the graph according to likely correlations, followed by another no-communication parallelizable stage for high confidence matching. A later general stage finishes the matching. This is a simplification of previous correlated matching techniques which required a complex interaction between general matching and reweighting the graph. Despite this simplification, which gives correlated matching a better chance of achieving real-time processing, we find the logical error rate practically unchanged. We validate the new algorithm on the fully fault-tolerant toric, unrotated, and rotated surface codes, all with standard depolarizing noise. We expect these techniques to be applicable to a wide range of other decoders.**

## 1 Introduction

The surface code [1, 2, 3, 4, 5, 6, 7, 8] is experimentally attractive as it requires only a 2D array of qubits with nearest neighbor interactions, and gate error rates of order 0.1% to achieve post-classical fault-tolerant quantum computation with fewer than 1M qubits [9]. Decoding the surface code must be done both accurately and in real time, and many algorithms have been proposed [10, 11, 12, 13, 14, 15, 16, 17].

Realizing a scalable, accurate, real-time surface code decoder remains an open problem. To move closer to this goal, we improve the algorithm used in [14] which performs a correlated-error version of minimum weight perfect matching [18, 19].

The method described by [14] required multiple runs of the matching decoder in order to take correlations between errors into account. Minimum weight perfect matching decoders operate on graphs of weighted edges, and the graphs had to be locally reweighted according to correlations in the underlying error model [14]. Reweighting triggered the erasure of partial solutions (matchings), leading to significant computational overheads – the entire matching had to be recomputed from scratch. Each reweighting operation had the overhead of a usual matching decoder $\mathcal{O}(d^6\log(d))$ [20] where $d$ is the distance of the surface code.

Our contribution is a pipelined method that significantly reduces the complexity and the execution time of the steps necessary for performing correlated decoding. Instead of re-running for an unknown number of times the full decoder of complexity $\mathcal{O}(d^6\log(d))$ in order to take reweights into account, we run a pre-matching decoder of complexity $\mathcal{O}(d^2)$ (Section A). The first decoding pipeline stage is pre-matching for reweighting, the second (optional) stage is pre-matching on the reweighted graph, and the third stage is the usual decoder using the pre-matched outputs and the reweighted graph. Practically, this works shows that fast correlated rematching is possible. Moreover, we avoid the significant overhead while maintaining decoding performance.

In Section 2.1, we describe how we extract and represent correlations in our circuits and error models. Section 2.2 describes a round by round algorithm to use the stream of detection events to infer correlations, reweight the detection graph, and perform a partial matching of detection events. This is followed by the algorithm

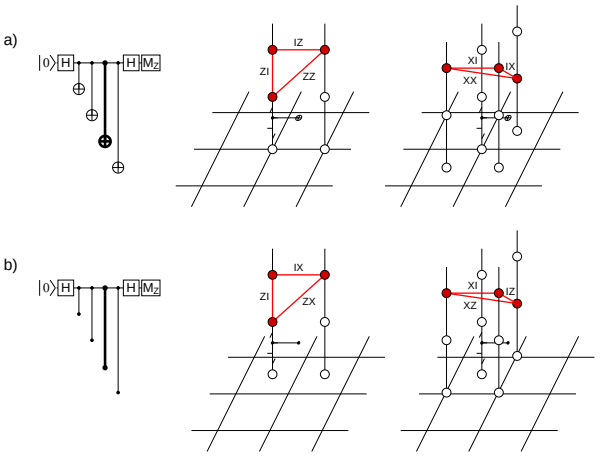


Figure 1: Three rounds of unrotated surface code: a) $X$ stabilizers measured on vertices; b) $Z$ stabilizers measured on faces. Time runs horizontally in the 2D and vertically in the 3D circuits. Bolded gates suffer all possible errors with only those leading to pairs of detection events (red) shown.

of [14, 17], with no further reweighting of the detection graph. Section 3 describes circuit level depolarizing noise simulations of the toric, unrotated, and rotated surface codes, all with and without correlated graph reweighting. The Appendix describes our pre-matching method, motivates its design for almost perfect parallelism and a simple method for ensure the correctness of the distributed/parallel algorithm.

## 2 Methods

We perform correlated reweighting and an initial partial matching in a strictly round by round manner, with no backtracking or revision, and without sacrificing performance in terms of the achieved logical error rate.

Our methods, including decoding by matching, is performed on a graph called *the detection graph*. The graph is generated by the analysis described in Section 2.1, and the graph edges are located in space-time (e.g. Fig. 1). Each edge has a probability $p$, and typically, it is more convenient to talk about the weight $w$ rather than their probability $p$.

Weights are associated to the edges in such a way that the most probable error corresponds to the minimum weight matching of the detection graph. Usually, most matching decoders use $w{=}-\ln\frac{p}{1-p}$ where $p$ is a probability of a single error associated with the qubit or gates of the code [2]. Technically, the edge probability is the probability of an odd number of independent errors occurring. In this work, we continue using $w{=}-\ln p$ similarly to how it was used in [14]. This weight approximation is sufficient for usual values of $p$, avoids the issue of negative weights [20], and achieves low logical error rates, as we shall see in Section 3.

### 2.1 Correlated pre-analysis of the surface code

To illustrate how an analysis of correlations in the surface code can be performed, consider a single two-qubit gate in multiple rounds of surface code stabilizer measurement. We assume depolarizing noise, but with each of the 15 nontrivial tensor products of $I, X, Y, Z$ potentially having a different probability. Each error on this gate can lead to anything from 0 to 4 detection events. Fig. 1 gives two examples of what can be observed.

Internally, for each gate we have a list of errors, and each error has a list of coordinates of detection events. Practically, our information is structured as follows *Gate* $\rightarrow$ *Errors* $\rightarrow$ *DetectionEventCoords*. Processing this information begins with identifying those errors that lead to: a) single detection events; b) pairs of detection events; c) 2+ detection events.

First, we focus on single detection events. These are represented graphically by an edge to an unspecified boundary. The coordinate of the generating detection event uniquely identifies each boundary edge. A single gate may generate no or many boundary edges. Each boundary edge keeps a list of errors that generated it, and each error on this gate that generates a boundary edge is appended to the appropriate error list.

Second, we focus on errors that lead to pairs of detection event. If both detection events are associated with boundary edges, we skip that error for the moment. Otherwise, we associate an edge with the detection event coordinates. As before, such an edge keeps a list of errors that generated it, and we append each generating error to the appropriate edge's error list.

Finally, we treat the 2+ detection events. The edges found so far form a basis, meaning all remaining errors generating 2+ detection events can be uniquely decomposed into two or more edges from this basis. Such errors are added to the error lists associated with each decomposed edge, and each copy of this error will be specially

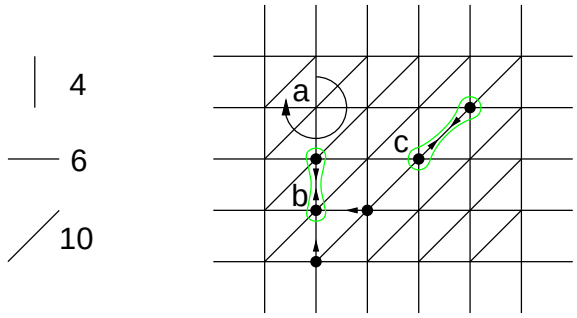


Figure 2: Pre-matching. Example of a graph with vertical edge weights 4, horizontal edge weights 6, and diagonal edge weights 8. a) Arbitrarily chosen ordering of the edges emanating from a vertex. b) Detection event with two lowest weight neighboring detection events, namely those vertically above and below it. Given the arbitrary ordering, the one above will be chosen. Note that since the detection event above has only a single neighbor, this choice will be mutual, indicated by a green bubble. c) Detection event with two equal lowest weight diagonal neighbors, the mutual chosen pair is shown.

annotated with the list of decomposed edges for later processing.

Some of the errors associated with each edge will have a list of decomposed edges. These decomposed edges, which were obtained from the 2+ detection events, form the basis of the correlated analysis. In the graph we have the information structured like *ParentEdge* ← *Error* ← *DecomposedEdges*. The decomposed edges that are different from the parent are called *correlated edges*. Each unique correlated edge is associated with a subset of *Errors* associated with the corresponding *ParentEdge*. The correlated pre-analysis operates as follows:

1. We analyze every gate in the computation, and generate a graph with each edge containing a list of errors. Each error will be labeled with a generating gate, and some of these errors containing a list of decomposed edges. This analysis can be done without excessive duplicate processing.

2. We calculate the total probability of each edge in the graph. The final edge probability $p_f$ is then approximated as simply $p_f{=}p_e{+}p_c$. We use the same expression, but with different parameter values, to compute $p_e$ (the edge probability) and $p_c$ (the correlated edge probability). See the following Section for more details.

$$p_e{=}\sum_i p_i \prod_{j\neq i}(1{-}p_j). \tag{1}$$

One could simply add up the probability of each error associated with the edge, however at higher error rates this is inaccurate and can lead to edge probabilities above 1. Instead, it is better to group the errors associated with an edge by gate, sum the probabilities of errors associated with each gate to give a list of probabilities $p_i$, then approximate the edge probability $p_e$ as the probability of exactly one independent error occurring.

## 2.2 Correlated reweighting and pre-matching

The pre-analysis generated many edges located in space-time, each with its own probability ($p_e$), and probability of being correlated with other nearby edges ($p_c$). The probability $p_c$ of each correlated edge can be calculated using Eq. 1 with appropriately reduced $p_i$ values. The relative probability of each correlated edge is divided by the edge probability $p_e$.

We present a heuristic for choosing highly likely edges in the detection graph that will be used to reweight it. We choose edges according to the following parallel pre-matching algorithm (see Fig. 2):

- For each detection event $e_0$, find the set of neighboring detection events $e_i$ with lowest weight connecting edges. If the set contains more than one, choose the one discovered first $e_1$ using any canonical ordering of the edges.

- For each detection event $e_0$ with a chosen neighboring detection event $e_1$, ask $e_1$ if $e_0$ is the chosen detection event of $e_1$. If yes, associate the two.

The first application of this algorithm will be called a "virtual" pre-matching, and will be used purely for detection graph reweighting. Given a virtual pre-matching, we can use the information about correlated edges derived in Section 2.1 to associate additional correlated probabilities $p_c$ with nearby edges. If more than one correlated probability is associated with a single edge, we permit the algorithm to have a race condition and only keep the last written value. The final edge probability $p_f$ is then approximated as

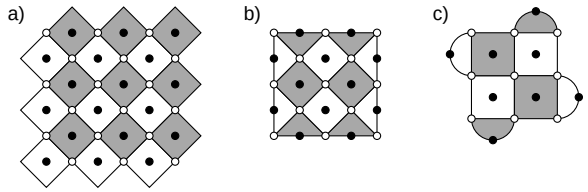


Figure 3: Distance 3 a) toric, b) unrotated, and c) rotated surface codes. Dark plaquettes represent $X$ stabilizers, light plaquettes represent $Z$ stabilizers. In all 3 cases the logical $X$ operator of interest runs from top to bottom.

simply $p_f{=}p_e{+}p_c$. This in turn becomes a new weight for the edge via $w{=}{-}\ln p_f$. Additional rounds of pre-matching are optional. Decoding can then be performed with the matchings kept from pre-matching time and passed on to the general matching algorithm, or other decoder.

## 3 Results

To illustrate the performance of our decoding algorithm, we will simulate three cases, the toric, unrotated, and rotated surface codes (Fig. 3), each with and without correlated graph reweighting. Simulation results can be found in Figs. 4–6.

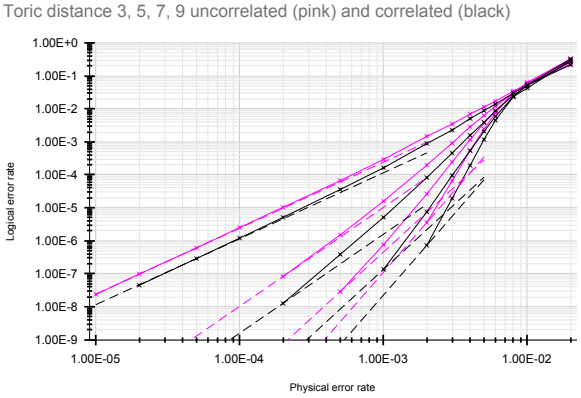


Figure 4: Toric uncorrelated (pink) and correlated (black) distance 3–9 simulations. Dashed lines show $p^2$, $p^3$, $p^4$ and $p^5$ lines respectively, the asymptotic slope of each line. The fact that the data curves are much steeper than the asymptotic curves for high distances and high gate error rates shows that logical errors are suppressed at even higher powers than these in this regime.

To the best of our knowledge, this is the first time these 3 commonly studied cases have been

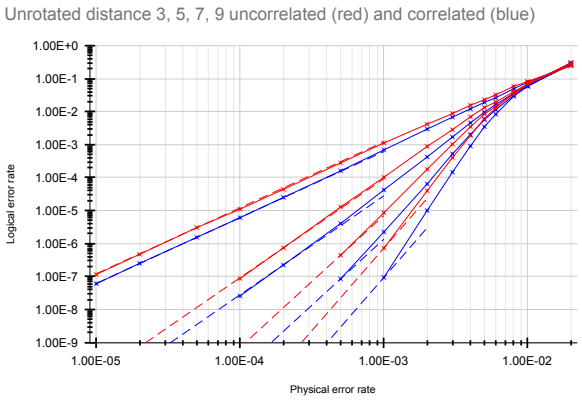


Figure 5: Unrotated uncorrelated (red) and correlated (blue) distance 3–9 simulations.

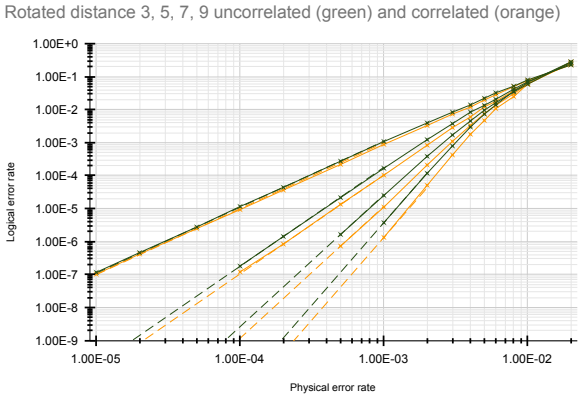


Figure 6: Rotated uncorrelated (green) and correlated (orange) distance 3–9 simulations.

simulated with a single framework with results gathered in one place for comparison. We will use standard equal gate duration 8-step CNOT-based circuits as shown in Fig. 1a, with all gates suffering standard depolarizing noise of equal probability $p$. Explicitly, initialization and measurement prepare and measure the wrong states with probability $p$, single-qubit gates including the identity suffer $X$,$Y$,$Z$ errors each with probability $p/3$, and CNOT gates suffer all 15 nontrivial tensor products of $I$,$X$,$Y$,$Z$ each with probability $p/15$.

We focus on the logical $X$ failure rate per round, and measure this by simulating a sufficiently large number of rounds $N$ such that the final probability of logical error is around 10%, then equate this with the probability of obtaining an odd number of logical errors in $N$ and back out the failure rate per round using a sum of binomial terms.

The simulations of Fig. 5 are indistinguishable from those reported in [14], with the distance 3 logical $X$ error in both cases just below $10^{-7}$ at a physical error $p$ of $10^{-5}$, and the distance 9 logical $X$ error also in both cases just below $10^{-7}$ at a physical error $p$ of $10^{-3}$.

## 4 Conclusion

A pipeline approach is a step towards real-time decoding of the surface code. We presented a method that operates on the stream of detection events and processes the data in a sequence stages implemented in a parallelizable manner that requires no communication.

The stages of the pipeline are: reweighting, pre-matching and full-matching. The functionality of the first two stages is based on a novel analysis of correlated errors. The correlated analysis is decoder agnostic, and the graph re-weighting highly hardware compatible. One can imagine measurements from a quantum computer streaming through dedicated hardware to generate a reweighted graph with optional partial matching that is then passed on to another decoder of the users choice.

Simulation results for the toric, rotated and unrotated surface code support the feasibility of the pipeline approach. Future work will focus on extending our results to other decoder types.

## Acknowledgments

AP was supported by Google Faculty Research Awards and a Fulbright Senior Researcher Fellowship.

## References

[1] S. B. Bravyi and A. Yu. Kitaev. "Quantum codes on a lattice with boundary" (1998). arXiv:quant-ph/9811052.

[2] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. "Topological quantum memory". J. Math. Phys. **43**, 4452–4505 (2002). url: https://doi.org/10.1063/1.1499754.

[3] Robert Raussendorf and Jim Harrington. "Fault-tolerant quantum computation with high threshold in two dimensions". Phys. Rev. Lett. **98**, 190504 (2007).

[4] R. Raussendorf, J. Harrington, and K. Goyal. "Topological fault-tolerance in cluster state quantum computation". New J. Phys. **9**, 199 (2007). url: https://doi.org/10.1088/1367-2630/9/6/199.

[5] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. "Surface codes: Towards practical large-scale quantum computation". Phys. Rev. A **86**, 032324 (2012). url: https://doi.org/10.1103/PhysRevA.86.032324.

[6] Austin G. Fowler and Craig Gidney. "Low overhead quantum computation using lattice surgery" (2019). arXiv:1808.06709.

[7] D. Litinski. "A game of surface codes: Large-scale quantum computing with lattice surgery". Quantum **3**, 128 (2019).

[8] Craig Gidney and Austin G. Fowler. "Flexible layout of surface code computations using autoccz states" (2019). arXiv:1905.08916.

[9] I. D. Kivlichan, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, Wei Sun, Zhang Jiang, N. Rubin, A. G. Fowler, A. Aspuru-Guzik, H. Neven, and R. Babbush. "Improved fault-tolerant quantum simulation of condensed-phase correlated electrons via trotterization". Quantum **4**, 296 (2020).

[10] Ruben S. Andrist, H. Bombin, Helmut G. Katzgraber, and M. A. Martin-Delgado. "Optimal error correction in topological subsystem codes". Phys. Rev. A **85**, 050302 (2012).

[11] Guillaume Duclos-Cianci and David Poulin. "Fault-tolerant renormalization group decoder for abelian topological codes" (2013). arXiv:1304.6100.

[12] Adrian Hutter, James R. Wootton, and Daniel Loss. "Efficient markov chain monte carlo algorithm for the surface code". Phys. Rev. A **89**, 022326 (2014).

[13] James Wootton. "A simple decoder for topological codes". Entropy **17**, 1946–1957 (2015).

[14] Austin G. Fowler. "Optimal complexity correction of correlated errors in the surface code" (2013). arXiv:1310.0863.

[15] P. Baireuther, M. D. Caio, B. Criger, C. W. J. Beenakker, and T. E. O'Brien. "Neural network decoder for topological color codes with circuit level noise". New J. Phys **21**, 013003 (2019). url: 10.1088/1367-2630/aaf29e.

[16] Nicolas Delfosse and Naomi H. Nickerson. "Almost-linear time decoding algorithm for topological codes". Quantum **5**, 595 (2021).

[17] Antonio deMarti iOlius, Patricio Fuentes, Román Orús, Pedro M. Crespo, and Josu Etxezarreta Martinez. "Decoding algorithms for surface codes" (2023). arXiv:2307.14989.

[18] Jack Edmonds. "Paths, trees, and flowers". Canadian Journal of Mathematics **17**, 449–467 (1965).

[19] J. Edmonds. "Maximum matching and a polyhedron with 0,1-vertices". J. Res. Nat. Bur. Standards **69B**, 125–130 (1965).

[20] Oscar Higgott. "Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching". ACM Transactions on Quantum Computing**3** (2022).

# A Appendix: Pre-matching

This section describes the method used during the analysis of correlated errors. Pre-matching is a best effort, very fast (low complexity) decoder that is not guaranteed to have a threshold or to be useful on its own for full error-correction. However, used within a decoding pipeline, pre-matching can inform later pipeline stages about the potential matchings existing in the detection graph. Pre-matching is used for computing the graph edges which should be reweighted.

Pre-matching uses only local vertex information without explicitly including a method for achieving a global optimum low weight matching. It is a greedy algorithm that assigns and updates three types of states to a detection event: zero-prematched (ZP), half-prematched (HP) and fully-prematched (FP). State transition rules are listed in Table 1 and discussed in Section A.2. Pre-matching operates the following steps:

- it initializes the states of the detection graph vertices to ZP;
- it iterates through all the vertices of the detection graph and updates their states based on the weights of the neighborhood edges (the neighboring vertex connected by an edge of minimum weight).

To this end an error detection graph like the one from Fig. 7 is used. Each graph vertex has an associated 3D coordinate of the type $(t,i,j)$ where $(i,j)$ are coordinates of the physical qubit that generated the detection event, and $t$ is an integer indicating time. Later detection events have higher values of the time coordinate $t$. Two examples are presented in Figs. 8 and 9. The temporal ordering of the detection events plays a role in how states are assigned and transformed.

The pre-matching algorithm has a linear complexity in the number of vertices in the detection graph. If the edges towards the neighboring detection events are not sorted based on their weights, the complexity becomes quadratic in the number of vertices, because we need to iterate through the edge list and select the one of minimum weight. The number of detection events scales quadratically with the distance $d$ of the code, and we use sorted edge lists, such that the total complexity of pre-matching is on the order of $\mathcal{O}(d^2)$.

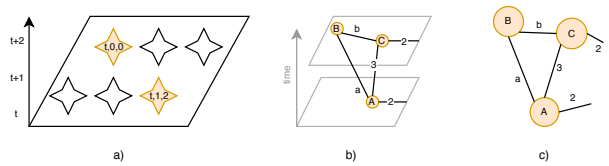


Figure 7: Pre-matching: a) Detection events have 3D coordinates, and are processed in the order of their time, row and column coordinates. For example, the first event to process is (1,0,0) and the second is (1,1,2). b) A graph of three detection events (A, B, C) is built and the weights of the edges are computed. In this example, only for the events A and C we consider the edges to the boundary of the code patch. c) The graph after eliminating the time layers visualization.

## A.1 Pre-matching condition (PMC)

We use the symbol $\leftrightarrow$ to indicate that two events A and B are FP, fully prematched. A full prematch between two events A and B works in two directions: A is prematched with B, iff B is prematched with A. We call *PMC* the event prematch condition that establishes one direction of

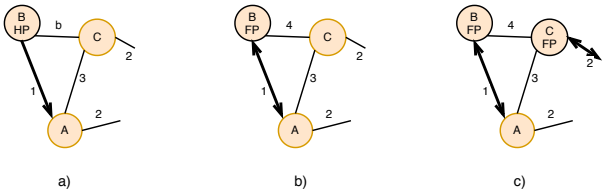


Figure 8: Advancing the state of the vertices. All vertices start in ZP (not illustrated) and the pre-match order is A, B, C. Example obtained by replacing $a$=1 and $b$=4 in Fig. 7c.: a) The lowest weight edge is the one connecting A to B, such that B is marked as half prematched HP stores a reference to A; b) After pre-matching B, the lowest weight edge points to A, and the state of B is updated to full prematched FP; c) The lowest edge weight of C is the one connecting to the boundary, and C is automatically fully prematched.

the prematching, either $\rightarrow$ or $\leftarrow$. The PMC of all the events is processed in the following order: a) increasing time $t$; b) increasing $i$-coordinate and c) increasing $j$-coordinate.

Two events A and B are fully prematched, the $A{\leftrightarrow}B$ relation exists, if $A{\rightarrow}B$ and $A{\leftarrow}B$. We use both directions of the arrows ($\rightarrow$ and $\leftarrow$) because we assume a temporal ordering between the detection events A and B (cf. Table 1). If the PMC is checked from A towards the future B then $A{\rightarrow}B$, otherwise if from a later B to a sooner A then $A{\leftarrow}B$.

A strict PMC can be formulated as $A{\rightarrow}B$ if B is the only low weight neighbor of A in the error detection 3D graph. The condition guarantees that the prematched events are also valid matchings from the perspective of minimum weight perfect matching. However, at higher physical error rates, the strict PMC is very seldom fulfilled, because multiple detection events of the same weight can exist at the same time in the neighborhood of another event. Thus, we introduce a relaxed PMC condition, such that more events are prematched without offering any guarantees that these form minimum weight matches.

Initially, all detection events are in the state ZP. Pre-matching is performed by iterating the detection events towards the future (increasing time coordinate). Assuming that pre-matching has reached vertex $A$, half-pre-matching is performed towards the future: if the event $B$ is later than $A$, and $A{\rightarrow}B$ then B will be in the HP state (if it was ZP) and will store a reference to $A$ (e.g. Fig. 8a). When it is the turn of B to be analysed for prematching, if $C{\leftarrow}B$ and C is the same as

coord(B) < coord(A)

| **A** | B | | |
|---|---|---|---|
| | ZP | HP | FP |
| ZP | ZP | E | ZP |
| HP | FP/ZP | E | E/ZP |
| FP | E | E | E |

coord(A) < coord(B)

| **B** | A | | |
|---|---|---|---|
| | ZP | HP | FP |
| ZP | HP | $\mathrm{ZP}_A$ | E |
| HP | E/ZP | $\mathrm{E/ZP}_{A,B}$ | E |
| FP | E | E | E |

Table 1: The state transformation rules when performing pre-matching starting from detection event A. Detection event B has the lowest weight from the neighborhood of A and is fulfilling the PMC. Depending on the coordinate of B, returned by the function *coord*, one of the two state transition tables is used. When not indicated by a subscript, the state of the latest (bold) detection event is updated.

the stored reference to A, then the state of B is transformed from HP to FP.

## A.2 Vertex state updates

Our pre-matching procedure updates the states of present and future detection events. If an event's state is ZP and it can be connected to the boundary, then the state is automatically updated to FP and the next detection event is considered (Section A.3). Otherwise, the state transformation procedure from the following paragraphs is applied.

The pre-matching procedure does not update states of detection events from the past. Considering that A is the detection event that is processed, and that the PMC function returns B, there are two possibilities:

- B has a coordinate lower than A;
- B has a coordinate higher than A.

For the first situation, $coord(B){<}coord(A)$ (cf. Fig. 9c where C is processed and $coord(A){<}coord(C)$), if B has state HP, then it is always an error, because B should have been already processed and its state is not valid. If the state of A is FP, this is impossible and always and error, because A is only now being processed. This leaves only four non-erroneous state transitions:

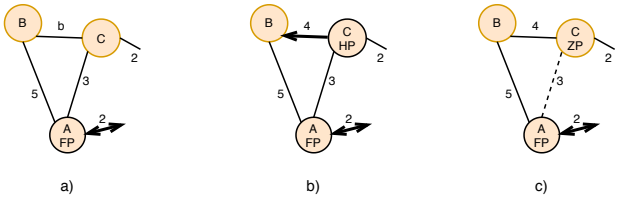


Figure 9: Reverting the state of the vertices. All vertices start in ZP (not illustrated) and the pre-match order is A, B, C. Example obtained by replacing $a$=5 and $b$=4 in Fig. 7c.: a) The lowest weight edge is connecting to the boundary, and A is automatically fully prematched. b) When pre-matching B, the lowest weight edge is towards C which is half prematched and will store a reference to B. c) Because A is FP and in the neighborhood of C, the lowest weight edge (2) is not considered and the next best option is pointing towards A (3), such that C cannot be prematched and its state is reset to ZP.

1. If A is ZP and B is ZP, then the state of A is not changed, because B is in the past.
2. If A is half-prematched HP and B is ZP, if the backwards reference from A points to B then both PMC directions are fulfilled and B is fully prematched and its state transitions to FP.
3. If the reference does not point to B, then the state of A is reset to ZP.
4. If A is HP but the B from the past is already fully prematched, then it is an obvious error if the reference from A points to B, and otherwise the state of A is reset to ZP.

In the second situation, $coord(A)<coord(B)$, it is always an error if A is FP. Also it is not possible for B to be FP, because it is in the future and could not have been processed by now (cf. Fig. 8a where A is processed). This leaves four possible configurations:

1. If both A and B are ZP, then the state of B is changed to HP and B will store a reference to A.
2. If A is HP and the future B is still ZP, it means that the backwards PMC for A is not fulfilled and the state of A is reset to ZP.
3. If B is HP and its reference points to A, it is definitely an error, because A is only now being processed and could have not changed the state of B to HP. Otherwise, if B is HP and A is ZP, then the state of A is kept ZP and the state of B is reset to ZP (this is a strict rule, the state of B could have been kept HP and be processed only later).
4. If B is HP and A is HP, the states of both A and B is reset to ZP.

The state transitions from in Table 1 are independent of the PMC. The pre-matching algorithm raises an error E each time it encounters an inconsistent (ie. incorrect) cases of vertex states. Inconsistencies are any situations where the following two rules do not hold.

1. before processing a vertex, the only valid states are ZP and HP (if it has been half-matched in the past). In other words, there is an error if the current node is already fully matched without having been processed first (the third row when $coord(A)<coord(B)$, and the third column when $coord(A)>coord(B)$);
2. after processing an vertex, the only valid states will be ZP and FP. In other words a processed vertex can be either not matched or fully matched, and nothing in between (e.g. HP);

Checking all the state transitions from Table 1 might seem trivial, but pre-matching is designed to be compatible with multi-threaded (ie. parallel, distributed) stream (ie. operations in the past are not allowed) processing. In such a setting, the detection graph is a resource shared and operated by all the threads. Data consistency and correctness needs to be ensured during the concurrent access of all the threads. Checking correctness is generally challenging when implementing parallel/distributed algorithms. However, our pre-matching algorithms is designed to be easily checked for correctness, and Table 1 is an exhaustive method of checking that parallel, multi-threaded pre-matching has been implemented and is operating correctly.

### A.3 Pre-matching with the boundary

The goal of pre-matching is to pair detection events and avoid where possible matching with the boundary (e.g. Fig. 9c). Pre-matching with the boundary is avoided as, in general, locally matching with the boundary seems to be the lowest weight choice, but when considering the

global sum of the weights this is often not the case.

Matching with the boundary is allowed whenever a detection event is processed and no other detection event in its neighborhood has the state FP (valid state from the past, cf. Table 1) or HP (valid state in the future, cf. Table 1). Considering two neighboring detection events A and B where both could be connected to the boundary we assume that such a decision will not result in a low weight global matching. For small distances, the ratio of events close to the boundary is higher than for larger distances. However, we consider that decoding should perform well for large distances and that our heuristic will result in *more realistic looking* matches.