Chapter 12

Matching

In this chapter, we look at *rounding*: a powerful technique to devise deterministic synchronous message-passing algorithms with polylogarithmic running time. The rough idea is to find a *fractional* solution to a problem and then turn this fractional solution into an *integral* one by rounding.

We illustrate the rounding technique with the example of *(maximal) matching.* The technique is way more general and has been applied to a variety of other problems including colorings and MIS.

Remarks (Maximal Matching in Low-Degree Graphs):

- Finding a maximal matching in a bipartite graph is particularly easy.

  Lemma 12.1. *A maximal matching in a 2-colored bipartite graph with maximum degree Δ can be found in $O(\Delta)$ time.*

  Proof. See Exercises. □

- Note that by Theorem 1.23 and Corollary 4.5, a MIS can be found in $O(\Delta + \log^* n)$ time in general (non-bipartite) graphs. Observing that a maximal matching in $G = (V, E)$ is nothing else than a MIS in the line graph (as observed in the remarks after Definition 4.19), we also have an algorithm with the same running time for the maximal matching problem. We can even show the following.

  Lemma 12.2. *Given a q-coloring of a graph with maximum degree Δ, a maximal matching can be computed in $O(\Delta + \log^* q)$ time.*

  Proof. See Exercises. □

Our goal though is to find deterministic algorithms with a running time that is polylogarithmic in n, no matter how large Δ is, trying to match the running time of the randomized counterpart, as given by Theorem 4.11.

Remark:

- Throughout this chapter, we make the simplifying assumption (without loss of generality) that Δ is a power of 2. All arguments would still be valid without this assumption but the notation would get a bit more tedious (e.g., writing $2^{-\lceil \log \Delta \rceil}$ instead of $\frac{1}{\Delta}$).

12.1 Matching

We start by formulating the matching problem as a linear program (i.e., a system of linear equations). Recall (from Definition 4.19) that a matching M is a subset of the edges such that no node has more than one incident edge in M. We can encode this set M with a variable $x_e \in \{0,1\}$ for every edge e where $x_e = 1$ if and only if $e \in M$. The condition for a node to not have more than one incident edge in the matching then can be written as

$$\sum_{e \in E(v)} x_e \leq 1$$

where $E(v) := \{e \in E \colon v \in e\}$ is the set of edges incident to v. This set of linear equations with integral values for x_e is called an *integer program* for matching where the goal is to maximize the number $|M| = \sum_{e \in E} x_e$ of edges in M:

$$\begin{aligned} &\max \sum_{e \in E} x_e \quad \text{s.t.} \\ &\forall e \in E \colon x_e \in \{0,1\} \\ &\forall v \in V \colon \sum_{e \in E(v)} x_e \leq 1 \end{aligned}$$

Relaxing the values to be non-integer, we get the following *linear program*:

$$\begin{aligned} &\max \sum_{e \in E} x_e \quad \text{s.t.} \\ &\forall e \in E \colon 0 \leq x_e \leq 1 \\ &\forall v \in V \colon \sum_{e \in E(v)} x_e \leq 1 \end{aligned}$$

Definition 12.3 (Fractional Matching). *We call a solution $(x_e)_{e \in E}$ to the above linear program a* fractional matching.

Definition 12.4 (Fractionality). *We call a fractional matching f-*fractional *if the smallest (non-zero) value of an edge is f for $0 < f \leq 1$. Note that a 1-fractional matching is an integral matching.*

Definition 12.5 (Approximation). *We say a (fractional or integral) matching $(x_e)_{e \in E}$ is c-approximate if $\sum_{e \in E} x_e \geq \frac{|M^*|}{c}$ for a maximum matching M^* and a number $c \geq 1$.*

For finding a maximal matching, an approximate maximum matching will turn out to be helpful. Indeed, there is a close connection between their sizes.

Lemma 12.6. *For a maximal matching M and a maximum matching M^* in a graph $G = (V,E)$, we have the following two properties:*

1. $|M| \geq \frac{|E|}{2\Delta - 1}$, *and*
2. $\frac{|M^*|}{2} \leq |M| \leq |M^*|$.

Proof. See Exercises. □

12.2 Approximate Fractional Matching

It turns out that computing an approximate fractional matching is straightforward in $O(\log \Delta)$ time, as we will discuss in this section.

Definition 12.7 (Loose/Tight Nodes/Edges)**.** *Given a fractional matching, we call a node v* loose *if $c_v := \sum_{e \in E(v)} x_e \leq \frac{1}{2}$ and* tight *otherwise. We call an edge* loose *if both its endpoints are loose and* tight *otherwise.*

Initially, we set $x_e = \frac{1}{\Delta}$ for all $e \in E$. This trivially satisfies the constraints $c_v = \sum_{e \in E(v)} x_e \leq 1$. Then, we iteratively raise the value of all loose edges (in parallel) by a factor 2 until all edges are tight.

Algorithm 12.8 Fractional Matching

```
Every edge $e$ executes the following code:
$x_e \leftarrow \frac{1}{\Delta}$
while both endpoints of $e$ are loose do
    $x_e \leftarrow 2 \cdot x_e$
end while
```

Remarks:

- This can be done in $O(\log \Delta)$ time, since at the latest when the value of an edge is $\frac{1}{2}$, both its endpoints are tight.
- If all edges incident to a node v double their value x_e, the value c_v doubles. Since only edges incident to loose nodes double their value, the value c_v of a node never exceeds 1, resulting in a valid matching.
- By construction, the fractional matching is (at least) $\frac{1}{\Delta}$-fractional and all edge values are in $\left\{\frac{1}{\Delta}, \frac{2}{\Delta}, \frac{4}{\Delta}, \frac{8}{\Delta}, \ldots, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\right\}$.
- If Δ is not a power of 2, we work with $2^{-(\lceil \log \Delta \rceil)}$ instead of $\frac{1}{\Delta}$.
- For simplicity, we formulate (most of) the algorithms in this chapter from the perspective of an edge instead of a node. It is an easy exercise to reformulate the algorithms for nodes.

Claim 12.9. *The fractional matching of Algorithm 12.8 is a 4-approximation.*

Proof. We prove the claim using a double counting argument (also called counting in two ways). Let every edge $e^* \in M^*$ in a maximum matching M^* distribute 1 dollar to the edges $e \in E$ such that every edge e receives at most $4x_e$ dollars. The claim follows since then $|M^*| \leq 4 \sum_{e \in E} x_e$.

An edge $e^* \in M^*$ has (at least) one tight endpoint, say v. It takes its dollar and distributes it among the edges $E(v)$ incident to v proportionally to their edge value x_e. Since $c_v = \sum_{e \in E(v)} x_e > \frac{1}{2}$, each edge in $E(v)$ receives at most twice its value. Now look at an arbitrary edge $e = \{u, v\}$. There is at most one edge incident to u and at most one edge incident to v in M^* (otherwise, M^* would not be a matching). Thus, e can have received at most $2x_e$ dollars from u and at most as much from v, thus in total no more than $4x_e$. □

Lemma 12.10. *Algorithm 12.8 computes a $\frac{1}{\Delta}$-fractional 4-approximate matching in $O(\log \Delta)$ time.*

12.3 Gradual Rounding in Bipartite Graphs

We now look at a fractional matching $(x_e)_{e\in E}$ as given by Algorithm 12.8. Recall that it (potentially) has edges with values in $\frac{1}{\Delta}, \frac{2}{\Delta}, \ldots, \frac{1}{2}, 1$. Our high-level plan is to gradually get rid of the fractional values by rounding them up by a factor 2 or down to 0. More concretely, we will first get rid of all edge values $\frac{1}{\Delta}$, then of all edge values $\frac{2}{\Delta}$, and so on. Intuitively, if in each of those $\log\Delta$ steps, we lose no more than a fraction $O\left(\frac{1}{\log\Delta}\right)$ of the matching size, we end up with a constant-approximate integral matching.

In the first step, we want to turn the $\frac{1}{\Delta}$-fractional matching into a $\frac{2}{\Delta}$-fractional matching $(x'_e)_{e\in E}$ such that we do not lose too much in the overall size. To that end, let us look at the graph induced by edges with value $f := \frac{1}{\Delta}$.

Definition 12.11 (Graph Induced by f-Edges)**.** *The graph $G_f = (V_f, E_f)$ is the subgraph consisting only of edges (and their endpoints) with value exactly f.*

Definition 12.12 (Rounding)**.** *Rounding G_f means identifying a subset of E_f whose edge values are raised from f to $2f$ (dropping the values of all other edges in E_f to 0) such that the resulting $2f$-fractional matching is still valid.*

Definition 12.13 (Perfect Rounding)**.** *A* perfect rounding *of G_f is a rounding of G_f such that for all nodes in v, half its incident edges are raised and the other half are dropped. See Figure 12.14.*

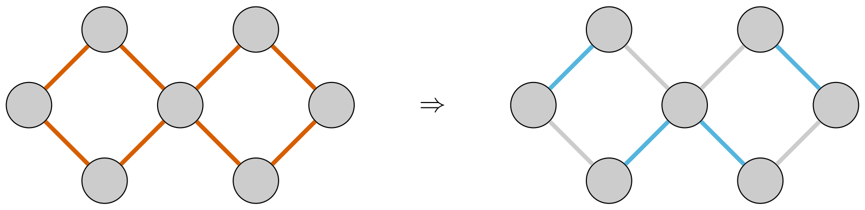

Figure 12.14: A perfect rounding of G_f where blue edges mean they have value f, orange stands for value $2f$, and grey for value 0.

Remarks:

- In a perfect rounding, the values c_v of all nodes v remain unchanged, i.e., $c'_v := \sum_{e\in E} x'_e = c_v$.
- In a perfect rounding, the overall size of the matching remains unchanged, i.e., $\sum_{e\in E} x'_e = \sum_{e\in E} x_e$.

Of course, a perfect matching would be our goal: rounding without losing anything in the overall matching size. But is that even possible (let alone computationally feasible)? Unfortunately, no. There cannot always be such a perfect rounding. Why? See Figure 12.15.

- If G_f has an odd-degree node, there is no perfect rounding.
- If G_f is a triangle (or, more generally, an odd-length cycle), there is no perfect rounding.

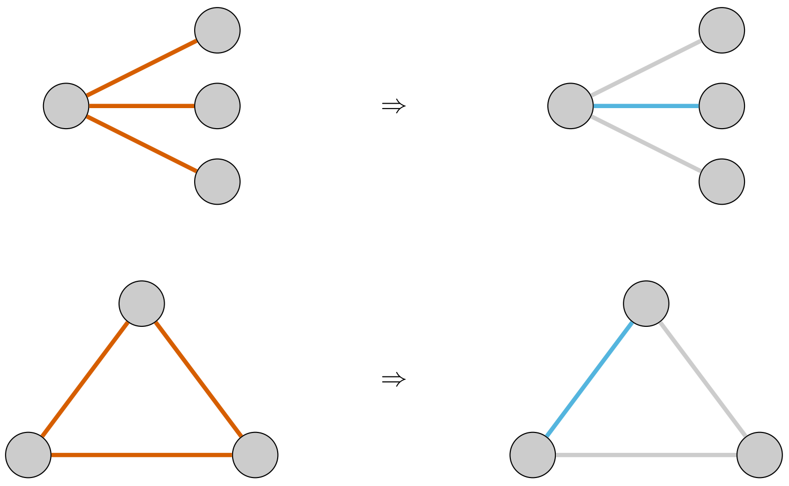

Figure 12.15: Two graphs G_f for which a perfect rounding is not possible: a graph with odd-degree nodes and a graph with an odd-length cycle.

In those cases, it is impossible to round half of a node's incident edges up and half of them down. While rounding more than half up runs the risk of destroying the matching property, rounding down only decreases the size of the matching. If in doubt, we thus always have to round down. This will result in a loss in the overall matching size, which luckily turns out to be not too large.

In this section, we will assume that the graph is bipartite (i.e., there are no odd-length cycles) and that the bipartition (that is, a 2-coloring of the nodes) is known. We will get rid of this assumption in the subsequent section.

We want to group incident edges of a node into pairs, in the hope that we can round one of each pair up and the other down. To this end, we look at the 2-decomposition G'_f of the graph G_f. See Figures 12.17 and 12.18 for examples.

Definition 12.16 (2-decomposition)**.** *The* 2-decomposition *of a graph looks as follows. For every node v, we introduce $\left\lceil \frac{\delta(v)}{2} \right\rceil$ copies and arbitrarily split its incident edges among these copies in such a way that every copy has degree 2, with the possible exception of one copy which has degree* 1 *(if v has odd degree).*

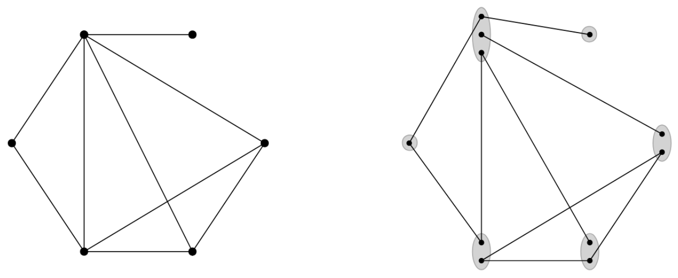

Figure 12.17: A (non-bipartite) graph and its 2-decomposition consisting of node-disjoint paths and cycles.

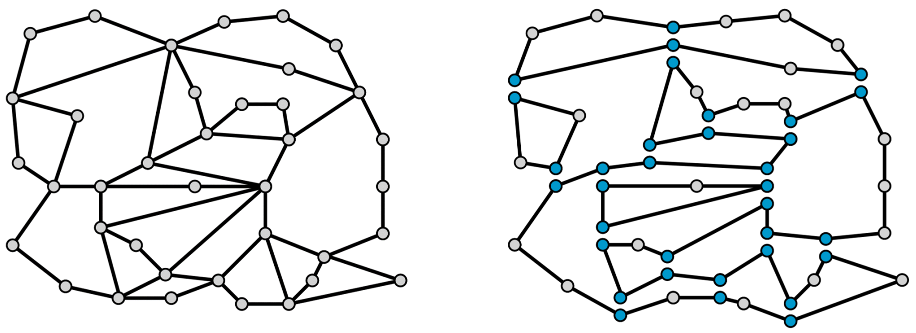
Figure 12.18: The 2-decomposition of a bipartite graph. Blue nodes are copies of the same original node, grey nodes are non-split original nodes.

Remarks:

- Since every node in a 2-decomposition has degree at most 2, it is a node-disjoint union of paths and cycles. If G is bipartite, all cycles have even length.
- Every node is part of $\lceil \delta(v)/2 \rceil$ cycles and at most one path. If every node in G_f has even degree, then G'_f consists of cycles only.

An almost perfect alternation of rounding up and rounding down for each path and cycle would result in an almost perfect rounding. But how do we compute that? We will discuss the rounding algorithm for cycles and paths separately, running in $O(\log \Delta)$ time each. We will have to distinguish short and long cycles/paths.

Definition 12.19 (Short/Long Paths/Cycles)**.** *A path or cycle is called short if its length is at most $\ell := 24 \cdot log\Delta$, and long otherwise.*

Rounding of Cycles

In a cycle, we would want the raise and drop of edge values to be alternating. Since the cycle has even length, the values $c_v = \sum_{e \in E(v)} x_e$ for all nodes v in the cycle would remain unaffected by this update. Moreover, the total value of the edges in the cycle would stay the same.

If the cycle is short, this perfect alternation can be identified in $O(\log \Delta)$ time as follows. All edges in the cycle agree on a global direction (one of the two) of the cycle. An edge is rounded up to $2f$ if it leads from a node with color 1 to a node with color 2 and rounded down to 0 otherwise (recalling that we are in a 2-colored graph). See Algorithm 12.20 and Figure 12.21.

Algorithm 12.20 Rounding Short Cycles

```
find globally consistent orientation of cycle
if $e$ goes from color-1-node to color-2-node then
    $x_e \leftarrow 2x_e$
else
    $x_e \leftarrow 0$
end if
```

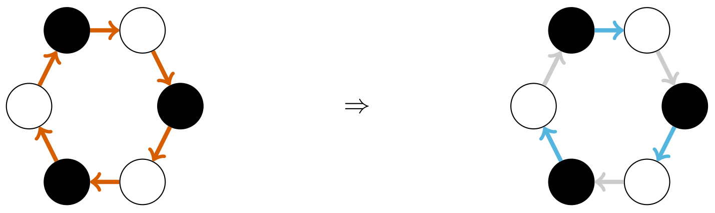

Figure 12.21: A perfect rounding of a 2-colored cycle with a given orientation of the cycle. An edge is rounded up (blue) if it leads from a node with color 1 (black) to a node with color 2 (white) and rounded down otherwise (grey).

For longer cycles, however, unfortunately, we cannot compute such a perfect alternation in $O(\log \Delta)$ time, since it is not possible for all edges to agree on the same direction of the cycle. Instead, we want only $O(\log \Delta)$-length chops of the cycle to agree on a direction. More specifically, our goal is to orient the edges in the cycle such that maximal directed paths have length at least ℓ.

Definition 12.22 (Maximal Directed Path)**.** *Consider a cycle with an orientation of each edge. A* maximal directed path *is a directed path in that cycle that cannot made any longer (since the two edges on the cycle incident to the boundary of the path are directed in the other direction). See Figure 12.23.*

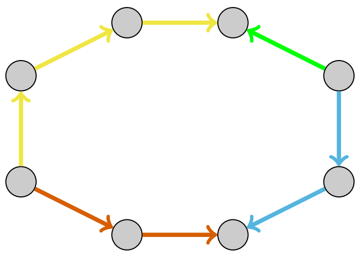

Figure 12.23: A cycle (with an arbitrary orientation of each edge) and its maximal directed paths indicated by different colors.

We can find an orientation of the edges in the cycle such that every maximal directed path has length at least ℓ as follows: We start with an arbitrary orientation of the edges (for instance, from color-1 nodes to color-2 nodes). For $i = 1, \ldots, \log(\ell)$, if there is a maximal directed path of length $< 2^i$, we merge it with the maximal directed path it points towards (where the arrows meet) by reversing all the edges of the shorter path, say, breaking ties arbitrarily.

Algorithm 12.24 Long Arrows

```
orient every edge $e$ arbitrarily
for $i = 1, \ldots, log(\ell)$ do
  if a maximal directed path has length $< 2^i$ then
    merge it with maximal directed path it is pointed towards
    by reversing shorter one (breaking ties arbitrarily)
  end if
end for
```

Remarks:

- Since we only merge two maximal directed paths that point towards each other, it cannot happen that a path P is trying to merge with a path P' (by reversing the edges of P) while at the same time P' is reversing its edges because it is trying to merge with a path P''. For each path, there is exactly one such potential merging partner and both paths agree on that!

- Initially, every maximal directed path has length at least 1. Merging two such paths leads to maximal directed paths of length at least 2, then 4, and so on. After iteration i, every maximal directed path has length at least 2^i, so after $\log(\ell)$ iterations, every maximal directed path has length at least ℓ.

- Since, in the end, maximal directed paths have length at least ℓ, at most a fraction $\frac{2}{\ell}$ of the edges are at the boundary of those paths.

- It is easy for an edge to identify whether it is a boundary edge or not: For $e = (u, v)$, if two arrows meet at v or if two arrows split at u, the edge is at a boundary.

After having computed an orientation with Algorithm 12.24, all edges at the boundary of a maximal directed path are rounded down (to avoid having a node with two incident edges being rounded up). With the remaining edges, we can proceed exactly as before (for short cycles): An edge directed from a node with color 1 to a node with color 2 is rounded up to $2f$; the other edges are rounded down. See Figure 12.25 and Algorithm 12.26. See Figure 12.27 for an overview of the different cases.

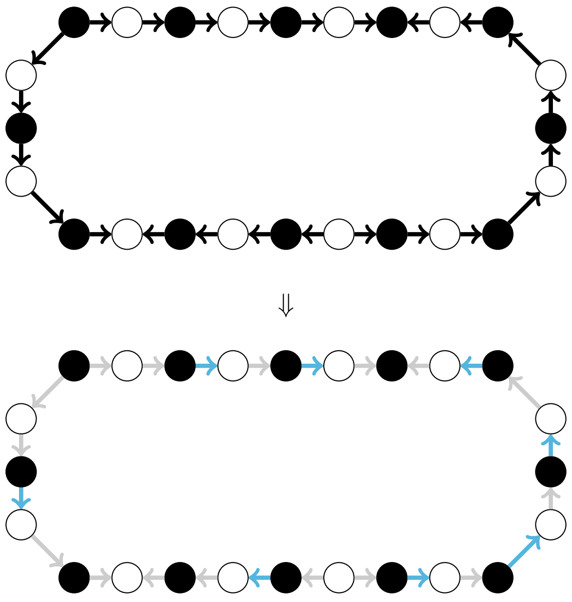

Figure 12.25: Rounding long cycles with long maximal directed paths.

Algorithm 12.26 Rounding Long Cycles

```
run Algorithm 12.24 (Long Arrows) to find orientation of each edge
if $e$ is a boundary edge or goes from color-2-node to color-1-node then
  $x_e \leftarrow 0$
else
  $x_e \leftarrow 2x_e$
end if
```

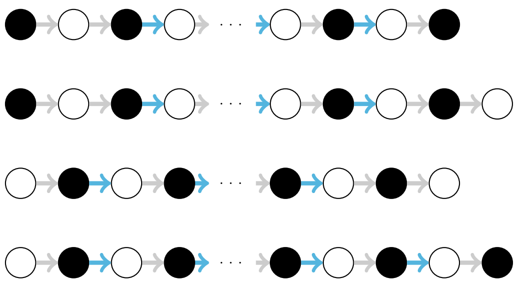

Figure 12.27: The four possible cases: Even-length and odd-length paths starting with a black and a white node, respectively.

In the worst case, a maximal directed path leads from a color-1 node to a color-2 node (see Figure 12.27, the second case). This results in the first two edges as well as the last two edges being rounded down with a perfect alternation starting and ending with rounding up in between. In this case, a value $3f$ is lost over a path that has total value at least $\ell \cdot f$. This thus results in a loss of at most a fraction $\frac{3}{\ell}$.

Claim 12.28. *Rounding long cycles leads to a loss* $\leq \frac{3}{\ell} \sum_{e \in E} x_e$.

Rounding of Paths

What to do about paths? First, observe that we can deal the same way with long paths as with long cycles (we anyway would not be able to tell the difference, would we?), resulting in a loss of at most a factor $\frac{3}{\ell}$, as argued in Claim 12.28.

For short paths, however, this approach does not work. Rounding down the boundary edges might result in the whole path being rounded down. For instance, think of a path of length 3 starting with a color-1 node. If the 2-decomposition consists of such paths only, the whole matching gets lost (we end up with a matching of size 0).

To find a middle ground between rounding the boundary edges down (possibly losing everything) and rounding the boundary edges up (possibly violating the matching condition), we proceed as follows. We first agree on a global orientation of the path (one of the two) so that we have a start node and an end node of the path. We round a boundary edge down if the boundary node is tight and round it up otherwise. The inner edges are alternating starting with a drop at the second edge. In other words, we round *even* edges down and *odd* edges up (if they are not at a boundary), where we call an edge *even* if it is the second, fourth, ... edge; and odd otherwise. See Algorithm 12.29 for the algorithm and Figure 12.30 for the possible constellations. Note that the 2-coloring is not relevant here.

Algorithm 12.29 Rounding Short Paths

```
find globally consistent orientation of path with start node $s$ and end node $t$
if $e$ is first edge then
  if $s$ is tight then
    $x_e \leftarrow 0$
  else
    $x_e \leftarrow 2x_e$
  end if
else if $e$ is last edge then
  if $t$ is tight then
    $x_e \leftarrow 0$
  else
    $x_e \leftarrow 2x_e$
  end if
else if $e$ is even edge then
  $x_e \leftarrow 0$
else
  $x_e \leftarrow 2x_e$
end if
```

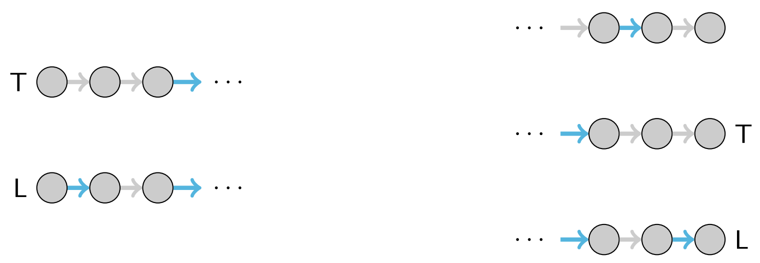

Figure 12.30: The possible constellations at the beginning of a path on the left: the start node can be tight (T) or loose (L). In the former case, the first edge is rounded down, in the latter it is rounded up. Starting from the second edge with a drop, there is a perfect alternation until before the end of the path. The possible constellations at the end of a path: If the path has even length, the last edge will be rounded down (in accordance with the perfect alternation) regardless of whether the end node is loose or tight. If the path has odd length, the last edge will only be rounded up if the end node is loose; otherwise, the last edge will be rounded down.

Observations:

- The worst case is an odd-length path with two tight endpoints. There, the first two as well as the last two edges are rounded down. The edges in between are alternating (starting and ending with a raise). This results in a loss of f at both boundaries. See Figure 12.31.
- For each node v, there is at most one copy in the 2-decomposition at the boundary of a short path (by construction). So a loss of f can happen at most once and only if v is tight (i.e., has value $c_v \geq \frac{1}{2}$).

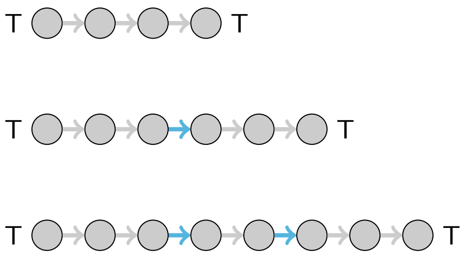

Figure 12.31: Bad cases of odd-length paths with two tight endpoints.

Claim 12.32. *Rounding short paths in G_f results in a loss $\leq 4f\sum_{e\in E} x_e$.*

Proof. Overall, by rounding short paths, we can lose at most f for every tight node. We therefore lose at most a fraction $\frac{f}{\frac{1}{2}} = 2f$ of a node's value, hence overall at most

$$2f\sum_{v\in V} c_v = 2f\sum_{v\in V}\sum_{u\in N(v)} x_{\{u,v\}} = 4f\sum_{e\in E} x_e$$

using the Handshaking Lemma (where $N(v)$ is the set of neighbors of v). □

One Rounding Step

In one rounding step—going from the f-fractional matching $(x_e)_{e\in E}$ to the $2f$-fractional matching $(x'_e)_{e\in E}$—we lose at most

$$\frac{3}{\ell}\sum_{e\in E} x_e$$

due to long paths and cycles (see Claim 12.28) and at most $4f\sum_{e\in E} x_e$ due to short paths (see Claim 12.32). Taken together, we have

$$\sum_{e\in E} x'_e \geq \left(1 - \frac{3}{\ell} - 4f\right)\sum_{e\in E} x_e,$$

summarized by the following lemma.

Lemma 12.33. *In the rounding step going from f-fractional to $2f$-fractional, the matching size decreases by a factor $\left(1 - \frac{3}{\ell} - 4f\right)$. This rounding takes $O(\log\Delta)$ time.*

Repeated Gradual Rounding

Now, we do not only want to apply the gradual rounding once but $\log\Delta$ times to get from $\frac{1}{\Delta}$-fractional to 1-fractional (i.e., integral).

Since the loss $4f$ becomes too large once f gets closer to 1, we cannot go all the way to integral with the above rounding strategy. Instead, we have to stop at a, say, $\frac{1}{16}$-fractional matching and resort to a different rounding strategy at that point.

Let us first see what matching size remains after having arrived there at a $\frac{1}{16}$-fractional matching by repeated rounding. We have

$$\begin{aligned}
\sum_{e\in E} x_e^{\left(\frac{1}{16}\right)} &\geq \left(1-\frac{3}{\ell}-\frac{1}{8}\right)\sum_{e\in E} x_e^{\left(\frac{1}{32}\right)}\\
&\geq \left(1-\frac{3}{\ell}-\frac{1}{8}\right)\cdot\left(1-\frac{3}{\ell}-\frac{1}{16}\right)\sum_{e\in E} x_e^{\left(\frac{1}{64}\right)}\\
&\geq \cdots\\
&\geq \left(1-\frac{3}{\ell}-\frac{1}{8}\right)\cdot\ldots\cdot\left(1-\frac{3}{\ell}-\frac{4}{\Delta}\right)\sum_{e\in E} x_e\\
&= \prod_{i=0}^{\log(\Delta)-5}\left(1-\frac{3}{\ell}-4\cdot\frac{2^i}{\Delta}\right)\sum_{e\in E} x_e
\end{aligned}$$

where $(x_e)_{e\in E}$ is the initial $\frac{1}{\Delta}$-fractional matching and $\left(x_e^{(f)}\right)_{e\in E}$ is used to denote the f-fractional matching in the rounding process. Intuitively, $\frac{3}{\ell}$ is in $O\left(\frac{1}{\log\Delta}\right)$, so over the $\log\Delta$ rounding steps, the expected loss due to this is a constant. On the other hand, the term $4\cdot\frac{2^i}{\Delta}$ grows geometrically and is dominated by the largest term, which is a constant. Overall, the loss thus is constant. This is formalized in the following claim.

Claim 12.34. *The resulting $\frac{1}{16}$-fractional matching is a constant factor smaller than the initial $\frac{1}{\Delta}$-fractional matching.*

Proof. Recalling that $\ell := 24\log\Delta$ from Definition 12.19, we observe that

$$\frac{3}{\ell}+4\cdot\frac{2^i}{\Delta}\leq\frac{3}{\ell}+\frac{1}{8}=\frac{1}{8\log\Delta}+\frac{1}{8}<\frac{1}{2}.$$

Using the inequality $1-y\geq e^{-2y}$ (known to be true for all $0\leq y\leq\frac{1}{2}$), we thus get

$$\begin{aligned}
\sum_{e\in E} x_e^{\left(\frac{1}{16}\right)} &\geq \prod_{i=0}^{\log(\Delta)-5} e^{-\left(\frac{6}{\ell}+8\cdot\frac{2^i}{\Delta}\right)}\sum_{e\in E} x_e\\
&= e^{-\sum_{i=0}^{\log(\Delta)-5}\left(\frac{6}{\ell}+8\cdot\frac{2^i}{\Delta}\right)}\sum_{e\in E} x_e = e^{-\sum_{i=0}^{\log(\Delta)-5}\left(\frac{6}{\ell}+8\cdot\frac{2^i}{\Delta}\right)}\sum_{e\in E} x_e.
\end{aligned}$$

Because

$$\sum_{i=0}^{\log(\Delta)-5}\frac{6}{\ell}=(\log(\Delta)-4)\cdot\frac{6}{24\log\Delta}\leq\frac{1}{4}$$

and

$$\sum_{i=0}^{\log(\Delta)-5} 8\cdot\frac{2^i}{\Delta}=\frac{1}{4}\sum_{i=0}^{\log(\Delta)-5}2^{-i}=\frac{1}{4}\cdot\frac{1-2^{-(\log(\Delta)-4)}}{1-2^{-1}}\leq\frac{1}{2},$$

we get

$$\sum_{e\in E} x_e^{\left(\frac{1}{16}\right)}\geq e^{-\frac{3}{4}}\sum_{e\in E} x_e,$$

concluding the proof. □

Lemma 12.35. *A constant-approximate* 16*-fractional matching can be computed in* $O(\log^2 \Delta)$ *in a* 2*-colored bipartite graph with maximum degree* Δ.

Proof. By Lemma 12.10, we get a constant-approximate $\frac{1}{\Delta}$-fractional matching in $O(\log \Delta)$ time. By applying the rounding procedure from Lemma 12.33 repeatedly for $O(\log \Delta)$ iterations, each iteration taking $O(\log \Delta)$ time, we end up with a $\frac{1}{16}$-fractional matching which is a constant factor smaller than the initial matching due to Claim 12.34. □

Final Rounding

Now that is not bad, but also not quite what we wanted. We are interested in an integral matching. So how do we turn the $\frac{1}{16}$-fractional matching from Lemma 12.35 into an integral one? The above rounding procedure will not work anymore: It could lead to the whole matching size getting lost.

Luckily, the graph $G' = (V', E')$ induced by edges with non-zero value looks pretty nice. Since every edge has value either at least $\frac{1}{16}$ or 0, a node cannot have more than 16 incident edges with non-zero value; the induced graph G' thus has maximum degree $\Delta' \leq 16$. By Lemma 12.1, in a 2-colored bipartite constant-degree graph, a maximal matching M can be found in $O(1)$ time, and, by Lemma 12.6 1., such a maximal matching has size

$$|M| \geq \frac{|E'|}{2\Delta' - 1} \geq \frac{\left|\left\{e \in E \colon x_e^{\left(\frac{1}{16}\right)} > 0\right\}\right|}{31} \geq \frac{\sum_{e \in E} x_e^{\left(\frac{1}{16}\right)}}{31},$$

where we use that $1 \geq x_e^{\left(\frac{1}{16}\right)}$ in the last inequality. Since the $\frac{1}{16}$-fractional matching is a c-approximate matching for some $c \geq 1$ due to Lemma 12.35, it follows $|M| \geq \frac{|M^*|}{c \cdot 31}$. In only $O(1)$ we thus have turned the almost-integral matching from Lemma 12.35 into an integral one, thus now know how to compute a constant-approximate integral matching in $O(\log^2 \Delta)$ time.

Lemma 12.36. *A constant-approximate matching can be computed in* $O(\log^2 \Delta)$ *time in a* 2*-colored bipartite graph with maximum degree* Δ.

12.4 Matching in General Graphs

The above rounding procedure does not work for non-bipartite graphs. In short odd-length cycles, a loss is inevitable. If the 2-decomposition consists of many short odd-length cycles, the loss is too big. Instead, our goal is to use the approximation algorithm for matchings in 2-colored bipartite graphs from Lemma 12.36 in a black-box manner to find approximate matchings in general graphs. We first explain how this can be done and then how the constant-approximation algorithm can be used to find a maximal matching, again as a black box.

Approximate Matching in General Graphs

The main idea is to transform the given general graph into a bipartite graph with the same edge set in such a way that a matching in this bipartite graph can be easily turned into a matching in the general graph. There is a well-known systematic approach to this, called *bipartite double cover*.

Definition 12.37 (Bipartite Double Cover). *The* bipartite double cover $B = (V_{in} \bigcup V_{out}, E_B)$ *of a graph* $G = (V, E)$ *is defined as follows. Let* $\overrightarrow{E}$ *be an arbitrary orientation of the edges* E*. Split every node* $v \in V$ *into two siblings* v_{in} *and* v_{out}*, and add an edge* $\{u_{out}, v_{in}\}$ *to* E_B *for every oriented edge* $(u, v) \in \overrightarrow{E}$*. Let* $V_{in} := \{v_{in} \colon v \in V\}$ *and* $V_{out} := \{v_{out} \colon v \in V\}$ *be the nodes with color 1 and 2, respectively. See Figure 12.38.*

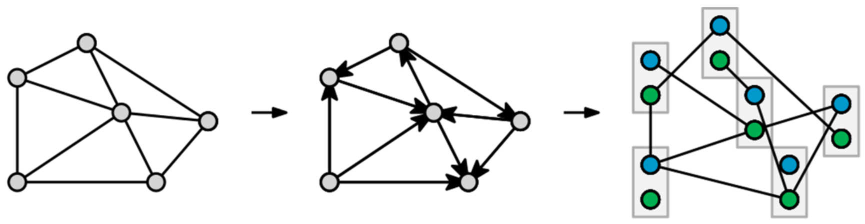

Figure 12.38: A graph and its bipartite double cover (for an arbitrary orientation). Here, the green color (color 1, say) is used for in-nodes and the blue color (color 2, say) for out-nodes.

Remarks:

- The bipartite double cover has twice as many nodes and the same number of edges as the original graph.
- The bipartite double cover is bipartite: Since it has edges only between in-siblings and out-siblings, any cycle must have an even length.
- A bipartite double cover can be computed in $O(1)$ time using an arbitrary orientation of the edges (e.g., from lower-ID to higher-ID).
- A matching in the bipartite double cover corresponds to a degree-2 subgraph in the original graph.

By Lemma 12.36, a c-approximate maximum matching M_B in the bipartite graph $B = (V_{\text{in}} \bigcup V_{\text{out}}, E_B)$ can be computed in $O\left(\log^2 \Delta\right)$ time for some $c \geq 1$. We now go back to V, that is, merge v_{in} and v_{out} back into v. This makes the edges of M_B incident to v_{in} or v_{out} now be incident to v, leaving us with a graph $G' = (V, M_B) \subseteq G$ with maximum degree 2.

We now compute a maximal matching M' in G' using the algorithm of Lemma 12.2, in $O(\log^* n)$ time with

$$|M'| \geq \frac{|M_B|}{3} \geq \frac{|M_B^*|}{3c} \geq \frac{|M^*|}{3c}$$

for some $c \geq 1$, a maximum matching M_B^* in B and a maximum matching M^* in G. Here, the first inequality follows from Lemma 12.6 1.; the second inequality follows from the fact that M_B is a c-approximate matching in B for some $c \geq 1$. For the last inequality, observe that a maximum matching M_B^* in B must be at least as big as a maximum matching M^* in G. This is because any matching in G certainly is also a matching in B (but not necessarily vice versa) as when going from G to B, we introduce additional nodes but leave the edge set unchanged (leading to fewer potential conflicts of an edge). Thus, M' is a $3c$-approximate maximum matching in G. See Figure 12.39 for an overview.

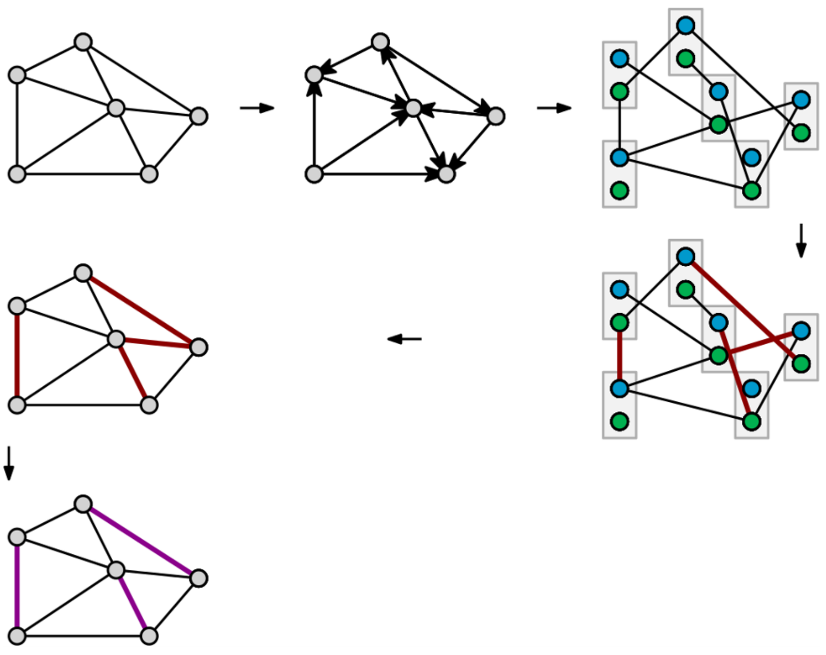

Figure 12.39: The algorithm step by step. We first orient the edges arbitrarily to compute the bipartite double cover. We then use our bipartite matching algorithm to compute a matching in this graph (red edges). By going back to the original graph, we end up with a degree-2 graph. In this graph, a maximal matching (purple) can be computed in $O(1)$ time, which results in a constant loss. Overall, this yields a constant approximation.

Lemma 12.40. *In a graph with n nodes and maximum degree Δ, a constant-approximate matching can be found in $O(\log^2 \Delta + \log^* n)$.*

Maximal Matching in General Graphs

We now show how to use the constant-approximation algorithm from above repeatedly in order to compute a maximal matching in a graph $G = (V, E)$ with maximum matching M^* of size $|M^*|$.

We use the algorithm from Lemma 12.40 to compute a c-approximate matching M in G in time $O(\log^2 \Delta + \log^* n)$ for some $c \geq 1$. We remove this matching M as well as all the matched nodes and all incident edges, resulting in a graph G_1. We observe that the maximum matching M_1^* in this graph G_1 has size at most $\left(1 - \frac{1}{c}\right)|M^*|$. Why?

Claim 12.41. *The maximum matching in G_1 has size $|M_1^*| \leq \left(1 - \frac{1}{c}\right)|M^*|$.*

Proof. Towards a contradiction, suppose G_1 has a matching M_1 of size $|M_1| > \left(1 - \frac{1}{c}\right)|M^*|$. Since the edges of G_1 (and hence M_1) are not incident to the edges in M (by construction of G_1), we have that $M \cup M_1$ is a valid matching in G. As M is a c-approximate matching in G, we have $|M| \geq \frac{|M^*|}{c}$, and thus $|M \cup M_1| > |M^*|$, contradicting the optimality of M^*. □

By repeatedly applying the above trick, after i iterations, the maximum matching in the remaining graph G_i has size $\left(1-\frac{1}{c}\right)^i |M^*|$. After $\log_{1-\frac{1}{c}} \frac{1}{n}$ iterations, the maximum matching size thus has decreased to $\frac{|M^*|}{n} < 1$, which means that the remaining graph is empty. This, on the other hand, means that the union of the matchings that have been removed in the previous iterations constitutes a maximal matching. See Figure 12.42 for an illustration.

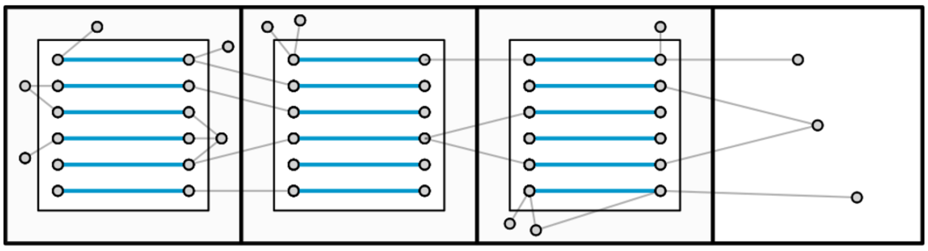

Figure 12.42: By repeatedly computing a matching (boxes with blue edges) in the remaining graph and removing matched nodes as well as their incident edges from the graph, we eventually end up with an empty remaining graph (depicted on the right). The union of all matchings is a valid matching (by construction) and is maximal (since the remaining graph is empty).

Theorem 12.43. *In a graph with n nodes and maximum degree Δ, a maximal matching can be found in $O(\log^2 \Delta \cdot \log n + \log^* n \cdot \log n)$ time.*

Remarks:

- Note that $\log_{1-\frac{1}{c}} \frac{1}{n} = \log_{\frac{c}{c-1}} n = O(\log n)$.
- The running time can be improved to $O(\log^2 \Delta \cdot \log n + \log^* n)$ by first precomputing a coloring and then using this coloring in each iteration. More concretely, we can use the algorithm of Linial [Lin92] to compute a $O\left(\Delta^2\right)$-coloring in $O(\log^* n)$ time.

Chapter Notes

The algorithm presented here is due to Fischer [Fis17, Fis20] and applies to many different variants of matching not discussed (e.g., weighted matching or b-matching). The idea of rounding has been introduced by Ghaffari et al. [GKM17] and since has been applied to a variety of problems, such as hypergraph maximal matching (and hence edge coloring, as well as maximal independent set in special graph classes) [FGK17], vertex coloring [GK22], set cover [FGG$^+$23] as well as MIS [GG23].

Many techniques go back to an earlier matching algorithm due to Hanckowiak, Karonski, and Panconesi [HKP01] in 2001, the first deterministic polylogarithmic algorithm for maximal matching (running in $O\left(\log^4 n\right)$ time). Before that, matching algorithms were only known for low-degree graphs [PR01]. The current best lower bound for maximal matching is the following: there is no deterministic algorithm that runs in $o\left(\Delta + \frac{\log n}{\log\log n}\right)$ [BBH$^+$19].

Bibliography

[BBH$^+$19] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 481–497. IEEE Computer Society, 2019.

[FGG$^+$23] Salwa Faour, Mohsen Ghaffari, Christoph Grunau, Fabian Kuhn, and Václav Rozhoň. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4409–4447. SIAM, 2023.

[FGK17] Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 180–191. IEEE, 2017.

[Fis17] Manuela Fischer. Improved deterministic distributed matching via rounding. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[Fis20] Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Computing*, 33(3):279–291, 2020.

[GG23] Mohsen Ghaffari and Christoph Grunau. Faster deterministic distributed mis and approximate matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1777–1790, 2023.

[GK22] Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1009–1020. IEEE, 2022.

[GKM17] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 784–797, 2017.

[HKP01] Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.

[Lin92] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1)(1):193–201, February 1992.

[PR01] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.