A Scaling Algorithm for Maximum Weight Matching in Bipartite Graphs*

Ran Duan
University of Michigan

Hsin-Hao Su
University of Michigan

Abstract

Given a weighted bipartite graph, the maximum weight matching (MWM) problem is to find a set of vertex-disjoint edges with maximum weight. We present a new scaling algorithm that runs in $O(m\sqrt{n}\log N)$ time, when the weights are integers within the range of $[0, N]$. The result improves the previous bounds of $O(Nm\sqrt{n})$ by Gabow and $O(m\sqrt{n}\log(nN))$ by Gabow and Tarjan over 20 years ago. Our improvement draws ideas from a not widely known result, the primal method by Balinski and Gomory.

1 Introduction

The input is a weighted bipartite graph $G = (V, E, w)$, where V consists of n left vertices and n right vertices, $|E| = m$, and $w : E \to \mathbb{R}$. A *matching* M is a set of vertex-disjoint edges. The *maximum weight matching (MWM)* problem is to find a matching M such that $w(M) = \sum_{e\in M} w(e)$ is maximized among all matchings, whereas the *maximum weight perfect matching (MWPM)* problem requires every vertex to be matched.

The MWPM problem and the MWM are reducible to each other [15]. To reduce from the problem of MWM to MWPM, obtain $\tilde{G}$ by making two copies of G and add a zero weight edge between each two copies of vertex. Then, $\tilde{G}$ is still bipartite and a MWPM in $\tilde{G}$ gives a MWM in G. Conversely, to reduce from the problem of MWPM to MWM, we simply add nN to the weight of each edge, where N is the maximum weight of the edges. This will guarantee that the MWM found is perfect.

Figure 1 shows the previous results on these problems. The first procedure for solving the MWPM problem dates back to 150 years ago by Jacobi [21]. However, the procedure was not discovered until recently [27]. In the 1950s, Kuhn [24] and Munkres [26] developed the "Hungarian" algorithm to solve the MWPM problem, where the former credited it to the earlier works of König and Egerváry. Later in [1], Balinski and Gomory gave an alternate approach to this problem, the primal method. The previous approaches grow the matching from empty while maintaining the feasibility of the dual program. In contrast, the primal method maintains the perfect matching from the beginning and fixes the infeasible dual solution along the way.

Later, Edmonds and Karp [7] and, independently, Tomizawa [32], observed that implementing the Hungarian algorithm for MWPM amounted to computing single-source shortest paths n times on a nonnegatively weighted graph. The running time of their algorithm depends on the best implementation of Dijkstra's algorithm, which has been improved over time [22, 9, 30, 31].

Faster algorithms are known when the edge weights are bounded integers in $[-N, N]$, and a word RAM model with $\log n + \log N$ word size is assumed. Gabow [11] gave a scaling approach for the MWPM problem, where he also showed the MWM problem can be solved in $O(Nm\sqrt{n})$ time. Gabow and Tarjan [15] improved the scaling approach to solve the MWPM in $O(m\sqrt{n}\log(nN))$ time. Later, Orlin and Ahuja [28] gave another algorithm with the same running time.

There are several faster algorithms for dense graphs. Cheriyan and Mehlhorn [2] exploited the RAM model and used a bit compression technique to implement Orlin and Ahuja's algorithm. Kao et al. [23] showed that the MWM problem can be decomposed into MWM problems with uniform weights, where a faster algorithm for the maximum cardinality matching problem in [8] can be applied. Extending from [25, 19], Sankowski gave an algebraic approach to solve this problem [29]. For general graphs, the relevant works are in [6, 10, 17, 12, 14, 13, 16, 25, 19].

In this paper, we look at the MWM problem with bounded integers in $[0, N]$, because negative weights can always be ignored. We present a new scaling algorithm that runs in $O(m\sqrt{n}\log N)$ time. Our algorithm improves the previous bound of $O(Nm\sqrt{n})$ by Gabow [11] and $O(m\sqrt{n}\log(nN))$ by Gabow and Tarjan [15]. Notice that our improvement is strict when $N = \omega(1)$ and $N = n^{o(1)}$. Other algorithms by [23] and [29] are not strongly polynomial and outperform ours only when $N = O(1)$ and the graph is very dense. The former re-

*This work is supported by NSF CAREER grant no. CCF-0746673 and a grant from the US-Israel Binational Science Foundation. H.-H. Su is partially supported by a fellowship from the Ministry of Education, R.O.C. (Taiwan).

Bipartite Weighted Matching

Year	Author	Problem	Running Time	Notes
1865 1955 1957 1964	Jacobi Kuhn Munkres Balinski & Gomory	MWPM	poly(n)	
1970 1971	Edmonds & Karp$^{(\star)}$ Tomizawa$^{(\star)}$	MWPM	$mn \log n$	Using binary heaps
1977	Johnson$^{(\star)}$	MWPM	$mn \log_d n$	Using d-ary heaps, $d = 2m/n$
1983	Gabow	MWPM MWM	$mn^{3/4} \log N$ $Nm\sqrt{n}$	N = max. *integer* weight
1984	Fredman & Tarjan$^{(\star)}$	MWPM	$mn + n^2 \log n$	Using Fibonacci heaps
1988 1992	Gabow & Tarjan Orlin & Ahuja	MWPM	$m\sqrt{n} \log(nN)$	*integer* weights
1996	Cheriyan & Mehlhorn	MWPM	$n^{2.5} \log(nN)(\frac{\log \log n}{\log n})^{1/4}$	
1999	Kao, Lam, Sung & Ting	MWM	$Nm\sqrt{n}(\frac{\log(n^2/m)}{\log n})$	*integer* weights
2002	Thorup$^{(\star)}$	MWPM	mn	*integer* weights, randomized
2003	Thorup$^{(\star)}$	MWPM	$mn + n^2 \log \log n$	*integer* weights
2006	Sankowski	MWPM	Nn^{ω}	*integer* weights, randomized ω = matrix mult. exponent
new result		MWM	$m\sqrt{n} \log N$	*integer weights*

Figure 1: Previous results on the MWPM and MWM problems. Algorithms that solve MWPM also solve MWM with the same running time. Conversely, algorithms that solve MWM can be used to solve MWPM, while the factor N becomes nN in the running time. (*) denotes implementations of the Hungarian algorithm using different priority queues.

quires $m = \omega(n^{2-\epsilon})$ for any $\epsilon > 0$, whereas the latter requires $m = \omega(n^{\omega-1/2})$, which is $\omega(n^{1.876})$ by the current fastest matrix multiplication technology [3].

Our approach consists of three phases. The first phase uses a search similar to one iteration of [15] to find a good initial matching. The second phase is the scaling phase. In contrast to [15], where they run up to $\log(nN)$ scales to ensure the solution is optimal, we run only up to $\log N$ scales. Then, the third phase makes the solution optimal by fixing the absolute error left by the first two phases. In some sense, our first and third phase have the effect equivalent to $0.5 \log n$ scales of the Gabow-Tarjan algorithm [15], thereby saving the additional $\log n$ scales. Like Balinski and Gomory's algorithm [1], our algorithm adjusts the matching throughout the second and third phases instead of finding a new one in each scale. In addition, as in [18], we bound our running time by using Dilworth's Lemma, in particular, that every partial order has a chain or anti-chain of size $\Omega(\sqrt{n})$.

1.1 Definitions and Preliminaries

A *matching* M is a set of vertex-disjoint edges. A vertex is *free* if it is not incident to an M edge, otherwise it is *matched*. If a vertex u is matched, denote its mate by u'. The MWM problem can be expressed as the following integer linear program, where x represents the incidence vector of a matching.

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} w(e)x(e) \\ \text{subject to} \quad & 0 \le x(e) \le 1,\ x(e) \text{ is an integer} \quad \forall e \in E \\ & \sum_{e=uv \in E} x(e) \le 1 \quad \forall u \in V \end{aligned}$$

It was shown that basic solutions to the linear program are integral. The dual of the linear program is as follows.

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V} y(u) \\ \text{subject to} \quad & y(e) \ge w(e) \quad \forall e \in E \\ & y(u) \ge 0 \quad \forall u \in V \\ \text{where we define} \quad & y(uv) \stackrel{\text{def}}{=} y(u) + y(v) \end{aligned}$$

By the complementary slackness condition, M and y are optimal iff $\forall e \in M$, $y(e) = w(e)$ and for all free vertices u, $y(u) = 0$. In the MWPM problem, the third inequality in the LP becomes equality, $\sum_{e=uv \in E} x(e) =$

$1, \forall u \in V$. Therefore, the condition $y(u) \geq 0, \forall u \in V$ is dropped in the dual program. If $\forall e \in M$, $y(e) = w(e)$, then M and y are optimal.

DEFINITION 1.1. *Given δ_0, let $\delta_i = \delta_0/2^i$, $w_i(e) = \delta_i \lfloor w(e)/\delta_i \rfloor$. The eligibility graph $G[c,d]$ at scale i is the subgraph of G containing all edges e satisfying either $e \notin M$ and $y(e) = w_i(e)$ or $e \in M$ and $w_i(e) + c\delta_i \leq y(e) \leq w_i(e) + d\delta_i$.*

An alternating path (or cycle) is one whose edges alternate between M and $E \backslash M$. Our algorithm consists of three phases, and we let $\delta_0 = 2^{\lfloor \log(N/\sqrt{n}) \rfloor}$ and the number of scales $L = \lceil \log N \rceil$, so that $w_L(e) = w(e)$ for all $e \in E$. An *augmenting walk/path/cycle* is defined differently in each phase:

1. Phase I: The phase operates at scale 0. An augmenting walk refers to an alternating path in $G[1,1]$ with free endpoints. For convenience, call such a path an augmenting path.
2. Phase II: The phase operates at scales $1 \ldots L$. An augmenting walk is either an alternating cycle in $G[1,3]$ or an alternating path in $G[1,3]$ whose end vertices have 0 y-values. For convenience, call the former an augmenting cycle and the latter an augmenting path. Notice that an endpoint of an augmenting path can be either free or matched. If an endpoint is matched, then we require its mate to be contained in the path as well.
3. Phase III: The phase operates at scale L. An augmenting walk is in $G[0,1]$ and defined the same as Phase II with one more restriction: The walk P must contain at least one matched edge that is not tight. That is, $y(e) \neq w_L(e)$, for some $e \in P \cap M$.

Given an augmenting walk P, by augmenting M along P, we get a matching $M \oplus P = (M \setminus P) \cup (P \setminus M)$. Given a subgraph $H \subseteq G$ and a vertex set $X \subseteq V$, let $V_{odd}(X, H)$ denote the set of vertices reachable through an odd-length alternating path in H starting with an **unmatched edge** that incidents to a vertex in X, and $V_{even}(X, H)$ be the set reachable via an even-length alternating path. For convenience, sometimes we denote a singleton $\{x\}$ by x. Let $\overrightarrow{G}$ denote the directed graph obtained by orienting edges e from left to right if $e \notin M$, from right to left if $e \in M$. Every alternating path in G must be a path in $\overrightarrow{G}$ and vice versa.

2 Algorithm

PROPERTY 2.1. *Throughout scale $i \in [0, L]$, we maintain matching M and dual variables y satisfying the following:*

1. ***(Granularity of y)*** *$y(u)$ is a nonnegative multiple of δ_i.*
2. ***(Domination)*** *$y(e) \geq w_i(e)$ for all $e \in E$.*
3. ***(Near Tightness)*** *$y(e) \leq w_i(e) + 3\delta_i$ for $e \in M$. At the end of scale i, it is tightened so that $y(e) \leq w_i(e) + \delta_i$ for $e \in M$.*
4. ***(Free Vertex Duals)*** *The y-values of free vertices are 0 at the end of scale i.*

LEMMA 2.1. *Let M^* be the optimal matching. If M and y satisfy Property 2.1 at the end of the scale L, then $w(M) \geq w(M^*) - n\delta_L$. Furthermore, when M is perfect and M^* is the optimal perfect matching, the same inequality holds if $y(e) \geq w(e)$ for all $e \in E$ and $y(e) \leq w(e) + \delta_L$ for $e \in M$.*

Proof.

$$\begin{aligned}
w(M) &= \sum_{e \in M} w(e) \\
&\geq \sum_{e \in M} y(e) - n\delta_L && \text{near tightness} \\
&= \sum_{u \in V} y(u) - n\delta_L && \text{free vertex duals} \\
&\geq \sum_{e \in M^*} y(e) - n\delta_L && \text{non-negativity} \\
&\geq \sum_{e \in M^*} w(e) - n\delta_L && \text{domination} \\
&= w(M^*) - n\delta_L
\end{aligned}$$

If M and M^* are perfect, then we can skip from the second line to the fourth line, since $\sum_{e \in M} y(e) - n\delta_L = \sum_{e \in M^*} y(e) - n\delta_L$.

The goal of each phase is as follows. Phase I finds the initial matching and dual variables satisfying Property 2.1 for scale $i = 0$. Phase II maintains Property 2.1 after entering from scale $i-1$ to i, for $i \in [1, L]$. In particular, we want to have $y(e) \leq w_i(e) + \delta_i$ for all $e \in M$ at the end of each scale. Phase III tightens the near tightness condition to exact tightness for all $e \in M$ after scale L so that $y(e) = w_L(e) = w(e)$ for all $e \in M$.

2.1 Phase I

In this phase, our algorithm will be working on $G[1,1]$ so that if one augments along an augmenting walk, all edges of the walk become ineligible.

Our algorithm maintains an invariant: All free left vertices, F, have equal and minimal y-values among left vertices and all free right vertices have zero y-values.

After the initialization, Property 2.1(4) is violated. We fix it by repeating the augmentation/dual adjustment steps until all vertices in F have zero y-values. The procedure described in the pseudocode is a modified Gabow-Tarjan algorithm [15] for one scale, where we always adjust dual variables by δ_0 in each iteration and stop when free vertices have zero y-values rather than when the matching is perfect.

Initialization:
$M \leftarrow \emptyset$.
Set $y(v) \leftarrow \begin{cases} \delta_0 \lfloor N/\delta_0 \rfloor & \text{if } v \text{ is a left vertex} \\ 0 & \text{otherwise} \end{cases}$.
repeat
  Augmentation:
  Find a maximal set P of augmenting paths in $G[1,1]$ and set $M \leftarrow M \oplus P$.
  Dual Adjustment:
  Let F be the left free vertices.
  For all $v \in V_{even}(F, G[1,1])$, set $y(v) \leftarrow y(v) - \delta_0$.
  For all $v \in V_{odd}(F, G[1,1])$ set $y(v) \leftarrow y(v) + \delta_0$.
until $F = \emptyset$ or $y(F) = 0$

After the augmentation step, there will be no augmenting paths in $G[1,1]$, which implies no free vertex is in $V_{odd}(F, G[1,1])$. Therefore, our invariant that right free vertices have zero y-values is maintained after the dual adjustment. Also, since all y-values of free vertices on the left will be decreased in every dual adjustment, they must be minimal among all left vertices. The number of augmentation/dual adjustment steps will be bounded by the number of total possible dual adjustments, which is $\delta_0 \lfloor N/\delta_0 \rfloor / \delta_0 \leq 2\sqrt{n}$. Thus, Phase I takes $O(m\sqrt{n})$ time.

In addition, the definition of eligibility on $G[1,1]$ ensures that if there exists $e \in M$ such that $y(e) = w_0(e) + \delta_0$ before the dual adjustment, then $y(e)$ cannot be increased after the adjustment. Therefore, $y(e) \leq w_0(e) + \delta_0$ for $e \in M$, near tightness is maintained throughout this phase. Likewise, if $e \notin M$ and $y(e) = w_0(e)$, then $y(e)$ does not decrease during the adjustment. Also, due to the definitions of V_{even} and an alternating path, $y(e)$ does not decrease for all $e \in M$. Thus, $y(e) \geq w_0(e)$ for all $e \in E$, domination is maintained throughout this phase.

2.2 Phase II At the beginning of scale $i \in [1, L]$, we set $y(u) \leftarrow y(u) + \delta_i$ for all left vertices u and do not change the y-values for all right vertices, so Property 2.1(2) (domination) is maintained. So is Property 2.1(3) (near tightness):

$$\begin{aligned} y(e) &\leftarrow y(e) + \delta_i \\ &\leq w_{i-1}(e) + \delta_{i-1} + \delta_i \\ &\qquad \text{by Property 2.1(3) at the end of scale } i-1 \\ &\leq w_i(e) + 3\delta_i \\ &\qquad \text{since } \delta_{i-1} = 2\delta_i \text{ and } w_{i-1}(e) \leq w_i(e) \end{aligned}$$

However, Property 2.1(4) may be violated, because now the y-values of left free vertices are δ_i. Hence, we will run one iteration of augmentation/dual adjustment step on $G[1,3]$ described in the pseudocode of Phase I to reduce them to zero. By the same reasoning in Phase I, domination and near tightness ($y(e) \leq w_i(e) + 3\delta_i, \forall e \in M$) will not be violated during the step, which implies Property 2.1 is now all maintained.

Next, we will repeat the augmentation/dual adjustment steps described in Section 2.2.1 and 2.2.2 on $G[1,3]$ until $y(e) \leq w_i(e) + \delta_i$ for all $e \in M$, or equivalently, until $M \cap G[2,3] = \emptyset$.

There are two reasons that we consider $G[1,3]$ rather than other definitions for eligibility. First, as in Phase I, since the definition of eligibility does not include matched tight edges, all edges of an augmenting walk become ineligible after we augment along it. Second, when doing the dual adjustment, we will not create any more matched edges in $G[2,3]$ (though they might be in $G[1,3]$), since the propagation of dual adjustments along the eligible edges e ensures that $y(e)$ will not be increased for $e \in M \cap G[1,3]$. This will be explained in Lemma 2.6.

2.2.1 Phase II - Augmentation When augmentation is called in Phase II, we need to eliminate all augmenting walks from the eligibility graph $G[1,3]$. This can be divided into two stages. In the first stage we eliminate the augmenting cycles, whereas in the second stage we eliminate the augmenting paths. Notice that unlike in Phase I, augmenting paths here may start or end with matched edges.

In the first stage, we will find a maximal set of vertex-disjoint augmenting cycles $\mathcal{C}$, which can be done by using a modified depth first search, *cycle_search*(x). We will inflict *cycle_search*(x) on every matched vertex x that has not been visited in previous searches. Recall that x' is the mate of x.

LEMMA 2.2. *The algorithm finds a maximal set of vertex-disjoint augmenting cycles $\mathcal{C}$. Moreover, if we augment along every cycle in $\mathcal{C}$, then the graph $G[1,3]$ contains no more augmenting cycles.*

Proof. Suppose the algorithm did not find a maximal set of vertex-disjoint augmenting cycles, let C be such a

Algorithm 1 *cycle_search*(u)

Mark u and u' as visited
for every unmatched edge $u'v$ **do**
  if v is visited and v is an ancestor of u in the search tree **then**
    Add the cycle consisting of the path from v to u' and the edge $u'v$ to $\mathcal{C}$.
    Back up the search until leaving *cycle_earch*(v) so the parent of v is on the top of the stack.
  else if v is not visited **then**
    Call *cycle_search*(v).
  end if
end for

cycle that is vertex-disjoint from all cycles in $\mathcal{C}$. Let $C = (v_1, v_2, \ldots v_k, v_1)$ so that v_1 is the vertex first entered in the search. Let t be the largest index such that v_t is visited by the search before the search backs up from v_1. Since v_t is not contained in any cycles in $\mathcal{C}$, the search must discover the next vertex of v_t in C. If $t < k$, then v_{t+1} is visited. If $t = k$, then we discovered a cycle containing the edge $v_k v_1$. Both cases lead to a contradiction.

Furthermore, if there exists a cycle C after augmentation, then this cycle must share a vertex v with some cycle $C' \in \mathcal{C}$ due to the maximality of $\mathcal{C}$. However, if v is contained in C, then C contains v and its mate. This contradicts the fact that there will be no eligible matched edge that incidents to v after the augmentation on C'.

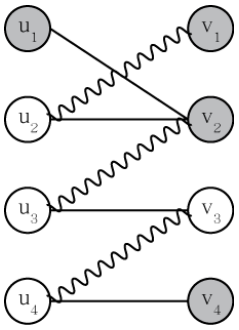

Figure 2: An example illustrating starting vertices and maximal augmenting paths in $G[1,3]$. The plain edges denote unmatched edges, while the curled ones denote matched edges. The shaded vertices denote vertices with zero y-values. Vertex u_1, v_1, and v_2 are starting vertices. The path $P = v_2u_3v_3u_4v_4$ is an augmenting path. However, it is not a maximal augmenting path, since either $u_1v_2u_3v_3u_4v_4$ or $v_1u_2v_2u_3v_3u_4v_4$ is an augmenting path containing P.

In the second stage, we will eliminate all the augmenting paths in $G[1,3]$. This is done by finding a maximal set of vertex-disjoint *maximal augmenting paths*. A maximal augmenting path is an augmenting path that cannot be extended to a longer one (see Figure 2). Note that we require such a path to be maximal, for otherwise it is possible that after we augment along a path, an endpoint of the path becomes free and is now an endpoint of another augmenting path.

Consider the graph $\overrightarrow{G}[1,3]$. It must be a directed acyclic graph, since $G[1,3]$ does not contain an augmenting cycle now. A vertex is said to be a *starting vertex* if it has zero y-value and it is either a left free vertex or a right matched vertex. Therefore, a starting vertex is a possible starting point of an augmenting path in $\overrightarrow{G}[1,3]$. Let S be the set of all starting vertices. We will initiate the search on every unvisited vertex in S in topological order of $\overrightarrow{G}[1,3]$. The way we initiate the search on x depends on whether x is free or not. If x is free, we will just call *path_search*(x). Otherwise, we will call *path_search*(x'). It is guaranteed that *path_search*(x) is called on left vertices.

Algorithm 2 *path_search*(u)

{Recall that x is the starting vertex and $\mathcal{P}$ is the maximal set of maximal augmenting paths we have found so far.}
Mark u as visited.
for every unmatched edge uv **do**
  if v is free {v is a right free vertex} **then**
    Add the path from x to v to $\mathcal{P}$ and terminate the search.
  else if v is not visited **then**
    Call *path_search*(v').
  end if
end for
if $y(u) = 0$ {u is a left matched vertex} **then**
  Add the path from x to u to $\mathcal{P}$ and terminate the search.
end if

If there exists an augmenting walk from x to v and v is not free, our search will explore the possibility that it can be extended from v before it is added to $\mathcal{P}$. If v is free, then it is impossible to extend the path. Furthermore, since we initiated the starting vertices in topological order, it is guaranteed that the path cannot be extended from x either. Therefore, the augmenting path found in our algorithm must be maximal.

LEMMA 2.3. *After we augment along every path in $\mathcal{P}$, the graph $G[1,3]$ contains no more augmenting paths.*

Proof. Suppose that there exists an augmenting path Q after the augmentation. Then by the maximality of

$\mathcal{P}$, there must be some augmenting path in $\mathcal{P}$ sharing vertices with Q. There can be two cases. Case 1: There exists $P \in \mathcal{P}$ and $v \in P \cap Q$ such that v is not an endpoint of either P or Q. In this case, by our definition of an augmenting path, P contains v and its mate before the augmentation on P and Q contains v and its mate after the augmentation. However, after the augmentation on P, there should be no eligible matched edge that incidents to v, thus Q cannot contain both v and its mate. Case 2: For all $P \in \mathcal{P}$, either $Q \cap P = \emptyset$ or Q and P intersect on their endpoints. Let P be the earliest path added to $\mathcal{P}$ such that P and Q intersect. Let x be the endpoint where they intersect, and x_P and x_Q be the other endpoints of P and Q. If $x_P = x_Q$ then there was an augmenting cycle, which is not possible. If x_P is a starting vertex, then *path_search*(x_P) should have found a longer augmenting path than P, since PQ is a longer one. On the other hand, if x is a starting vertex, it must be a right matched vertex and becomes free after augmentation, so x_Q must also be a starting vertex. Since our search is called in topological order on starting vertices, x_Q must be called before x, which implies that the first augmenting path found that intersects Q contains x_Q but not x.

2.2.2 Phase II - Dual Adjustment Let B be the set of violated matched edges that need to be tightened before the end of scale i, that is, $B = \{e \in M : y(e) - w_i(e) > \delta_i\} = G[2,3] \cap M$. Define the *badness*, $f(e)$, to be the amount edge e has violated. That is, $f(e)$ is $(y(e) - w_i(e) - \delta_i)/\delta_i$ for $e \in B$, $f(e)$ is 0 for $e \notin B$. Let $f(B) = \sum_{e \in B} f(e)$ be the total badness of B. Then B is empty if and only if $f(B) = 0$, since $f(e) > 0$ for $e \in B$. The goal of dual adjustment is to tighten Property 2.1(3), namely, to decrease $f(B)$ to 0.

A $B' \subseteq B$ is said to be a *chain* if there is an eligible alternating path containing B'. On the other hand, B' is said to be an *anti-chain* if for any $m_1, m_2 \in B'$ such that $m_1 \neq m_2$, there exists no eligible alternating path containing them.

LEMMA 2.4. *For any $t > 1$, there exists $B' \subseteq B$ such that either B' is a chain with $f(B') \geq \lceil t \rceil$ or B' is an anti-chain with $|B'| \geq \lceil f(B)/2t \rceil$. Moreover, such B' can be found in linear time.*

Proof. This lemma basically follows from Dilworth's Lemma [5]. First obtain $\overrightarrow{G}[1,3]$ by orienting the edges in $G[1,3]$ and assign the length to be $f(e)$ for every $e \in \overrightarrow{G}[1,3]$. Then, $\overrightarrow{G}[1,3]$ must be a directed acyclic graph since we have no augmenting cycles.

Let S denote the vertices with zero in-degrees. Compute the longest path from S to every vertex in $\overrightarrow{G}[1,3]$, which can be done in linear time in topological order. If there exists a path P having length at least $\lceil t \rceil$, then $P \cap B$ must be a chain with $f(P \cap B) \geq \lceil t \rceil$. Otherwise, for every $uv \in B$ (assume that v is the left vertex), the length of the longest path from S to v is in the range of $[1, \lceil t \rceil - 1]$. Since $f(e) \leq 2$ for $e \in B$, we must have at least $\lceil |B|/t \rceil \geq \lceil f(B)/2t \rceil$ such v having the same longest distance from S. If the distance is k, then the set $B' = \{uv \in B : v$ is a left vertex and the longest distance from S to v is $k\}$ must be an anti-chain. For $u_1v_1, u_2v_2 \in B$, if there is an alternating path containing them in $G[1,3]$, there must be a path from u_1v_1 to u_2v_2 or from u_2v_2 to u_1v_1 in $\overrightarrow{G}[1,3]$ so the longest distance from S to v_1 and v_2 must be different.

Below we show that if B' is a chain we can decrease the total badness by $f(B')$ in linear time. On the other hand, if B' is an anti-chain, then we can decrease the total badness by $|B'|/2$ also in linear time.

2.2.3 Phase II - Dual Adjustment - Anti-chain Case

DEFINITION 2.1. *A vertex x is said to be dual adjustable if for every $v \in V_{odd}(x, G[1,3])$, v is not free and for every $v \in V_{even}(x, G[1,3])$, $y(v) > 0$.*

LEMMA 2.5. *For every $e = uv \in B$, either u is adjustable or v is adjustable or both. Furthermore, all adjustable vertices can be found in $O(m)$ time.*

Proof. First, if $e = uv \in B$ and u and v are both not adjustable, then by our definition of adjustable, there exist vertices w and x having zero y-values where $w \rightsquigarrow u \to v \rightsquigarrow x$ is an augmenting path. However, this contradicts the fact that there are no augmenting paths after the augmentation step.

To find the adjustable vertices, it is rather convenient to mark up all those unadjustable vertices. Let $\tilde{V} = \{v : v$ is free or v is matched and $y(v') = 0\}$, and mark all vertices as unadjustable in $V_{odd}(\tilde{V}, G[1,3])$. This can be done in linear time.

Let $B' \subseteq B$ be an anti-chain. We call *antichain_adjust*(B') to adjust the dual variables. In the procedure, we will pick a set of dual adjustable vertices X that are adjacent to B' and on the same side, then do a dual adjustment starting at X. Since by Lemma 2.5, for any $e = uv \in B'$ either u is adjustable or v is adjustable or both, we can guarantee that $|X| \geq |B'|/2$. See Figure 3 for an example.

LEMMA 2.6. *The dual adjustment starting at X will not break Property 2.1(1), 2.1(2), or 2.1(4). Furthermore, it makes Property 2.1(3) tighter by decreasing $f(B)$ by $|X|$.*

Algorithm 3 *antichain_adjust*(B')

Let $\tilde{V} = \{v : v$ is free or v is matched and $y(v') = 0\}$.
Mark vertices in $V \setminus V_{odd}(\tilde{V}, G[1,3])$ as adjustable vertices.
Let $X_L = \{u : uv \in B'$ and u is a left adjustable vertex $\}$.
Let $X_R = \{u : uv \in B'$ and u is a right adjustable vertex$\}$.
If $|X_R| > |X_L|$, then let $X = X_R$; otherwise let $X = X_L$.
Dual adjustment starting at X:
For all $v \in V_{even}(X, G[1,3])$, set $y(v) \leftarrow y(v) - \delta_i$.
For all $v \in V_{odd}(X, G[1,3])$, set $y(v) \leftarrow y(v) + \delta_i$.

Proof. Since every vertex in X is adjustable, every vertex $v \in V_{odd}(X, G[1,3])$ must have $y(v) > 0$, implying $y(v)$ will be non-negative after subtracting δ_i. Thus, Property 2.1(1) is maintained. In addition, by the definitions of an alternating path and V_{even}, $V_{even}(X, G[1,3])$ cannot contain a free vertex. Therefore, no dual variables of free vertices are adjusted, meaning Property 2.1(4) is maintained. Since all vertices in X are on the same side, $y(e)$ can change by at most δ_i. We only need to check:

1. If $e = uv$ is tight before the adjustment, Property 2.1(2) (domination) holds for e after the adjustment: If $e \notin M$, then e is eligible. If the y-value of an endpoint gets subtracted by δ_i then another endpoint must be added by δ_i, which means $y(e)$ does not decrease. If $e \in M$, then it is not possible for u or v to be in $V_{even}(x, G[1,3])$, since e is ineligible and we start with an unmatched edge. Therefore, domination holds on e after the adjustment.

2. $f(B)$ decreases by $|X|$: If e is tight before the adjustment, then increasing $y(e)$ by δ_i contributes nothing to $f(B)$. If e is not tight, then e is eligible and $f(e)$ cannot be increased either, since if one endpoint gets added by δ_i, then another endpoint must be subtracted by δ_i. Furthermore, if $e \in B'$ and e is incident to a vertex in X, then one endpoint of e is in $V_{even}(X, G[1,3])$ and the other cannot be in $V_{odd}(X, G[1,3])$, because B' is an anti-chain. Therefore, $f(e)$ decreases by exactly 1.

Therefore, by doing the dual adjustment starting at X, we can decrease $f(B)$ by at least $|B'|/2$.

2.2.4 Phase II - Dual Adjustment - Chain Case

In the chain case, there exists an alternating path containing B'. Take P to be the minimal alternating path containing B' so that P starts and ends with edges in B'. If we augment along P, then the edges in B' no longer contribute to $f(B)$ since they become unmatched, and new M-edges contribute nothing to $f(B)$. However, the endpoints of P, say u and v, become free while possibly having positive y-values. Hence we will need to make them matched by augmentation or decrease their y-values to zero. In this subsection, we relax our definition of augmenting path such that the y-value of each endpoint is 0 **except** if it is u or v. We perform a search similar to Phase I on u until an augmenting path P_u starting from u is found or $y(u)$ becomes zero (which is a degenerated case when $P_u = \{u\}$). After the search, we will not augment P_u immediately but perform another search again on v to find an augmenting path P_v. Now if there exists an augmenting path Q in $G[0,3]$ whose endpoints are u and v, then we will augment along it. See Figure 4 for an example. Otherwise, we let $Q = P_u \cup P_v$ and then augment along Q. In this case, we must have $P_u \cap P_v = \emptyset$, for otherwise an augmenting path in $G[0,3]$ between u and v exists. In the searches, we will use $G[0,3]$ as the eligibility graph, which ensures the weight

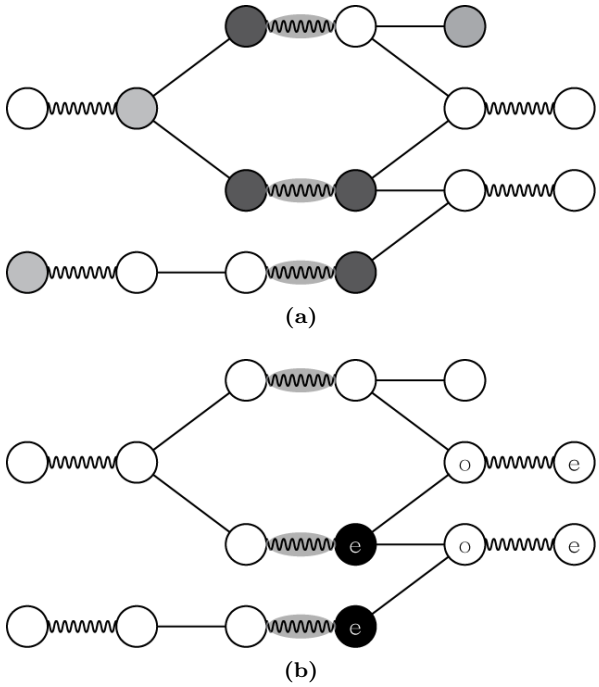

Figure 3: An example of an eligible graph that illustrates an anti-chain and adjustable vertices. (a) The light shaded vertices denote vertices with zero y-values. The shaded matched edges form an anti-chain of size 3. The dark shaded vertices are adjustable vertices of the anti-chain. (b) The dark vertices denote X, the selected vertices for the dual adjustment. Vertices marked with 'e' and 'o' denote vertices in $V_{even}(X, G[1,3])$ and $V_{odd}(X, G[1,3])$ respectively.

Algorithm 4 *search*(x)

If x is a right vertex, set $\overrightarrow{G} \leftarrow \overrightarrow{G}^T$ (reverse the edges).

For each $e \in \overrightarrow{G}$, assign a new weight $w'(e) = \begin{cases} y(e) - w_i(e) & \text{if } e \notin M \\ 0 & \text{if } e \in M \end{cases}$

Compute the distance $d(z)$ from x to z for every $z \in \overrightarrow{G}$, where $d(z) = \infty$ if z is not reachable from x.

Let $h(z) = \begin{cases} d(z) & \text{if } z \text{ is free and not on the same side as } x \\ d(z) + y(z) & \text{if } z \text{ is on the same side as } x \\ \infty & \text{otherwise} \end{cases}$.

Let z_{min} be the vertex such that $h(z)$ is minimum, and let $\Delta = h(z_{min})$.

Let P_x be the shortest path from x to z_{min}.

Set $y(z) \leftarrow \begin{cases} y(z) - max\{0, \Delta - d(z)\} & \text{if } z \text{ is on the same side as } x \\ y(z) + max\{0, \Delta - d(z)\} & \text{if } z \text{ is not on the same side as } x \end{cases}$

return P_x

of the new matching we get does not decrease. Below we describe how the search works.

Let $x \in \{u, v\}$ be the free vertex that we perform the search on. If there exists an augmenting path in $G[0,3]$ starting at x, then we will stop. Recall that the other endpoint of x can be either free or matched. On the other hand, if there is no augmenting path, then let γ be the minimum of $min\{y(z) : z \in V_{even}(x, G[0,3])\}$ and $min\{y(v_1v_2) - w_i(v_1v_2) : v_1 \in V_{even}(x, G[0,3])$ and $v_2 \notin V_{odd}(x, G[0,3])\}$. Then, add γ to the y-value of every vertex in $V_{odd}(x, G[0,3])$ and subtract γ from vertices in $V_{even}(x, G[0,3])$. Keep repeating the adjustment until we find an augmenting path starting at x. Similar to one iteration in the Hungarian algorithm, this process is equivalent to computing shortest paths from x, which is described in Algorithm 4, *search*(x).

In *search*(x), Δ is the amount of dual adjustment needed before an augmenting path opens up. The augmenting path starts from x and ends at some z_{min}, where z_{min} can be either a free vertex on the opposite side of x or a zero y-valued matched vertex on the same side as x. For the former situation, the dual adjustment needed is $d(z_{min})$. For the latter situation, we not only need to reach z_{min} but also need to decrease its y-value to 0, so the dual adjustment needed is $d(z_{min}) + y(z_{min})$. After finding Δ, we will adjust the dual variables accordingly. *search*(x) returns an augmenting path P_x starting at x.

By Property 2.1(1), $d(z)$ must be a non-negative multiple of δ_i. Since our goal of computing the shortest path is to find Δ, we can just compute those $d(z)$ which are no more than Δ. This can be done in $O(m + \Delta/\delta_i)$ time by using an array as a priority queue in Dijkstra's algorithm. (See Dial's implementation [4].)

LEMMA 2.7. *Augmenting along P and then Q does not decrease the weight of the matching and $\Delta_u + \Delta_v \leq 3n\delta_i$, where Δ_u and Δ_v are the amount of dual adjustments done in search(u) and search(v). Thus, the search can be done in $O(m)$ time.*

Proof. Suppose M is the original matching, M' is the matching obtained by augmenting along P, and M'' is the final matching after augmenting along Q. Let $w''(e) = y(e) - w_i(e)$ (notice that w'' differs from w' on the matched edges). For a quantity q denote its value before both searches by q_{old} and after both searches by q_{new}. After the searches, we must have:

$$\begin{aligned} w_i(Q \setminus M') &= \sum_{e \in Q \setminus M'} y_{new}(e) \\ &\qquad \text{tightness on unmatched edges} \\ &= y_{new}(u) + y_{new}(v) + \sum_{e \in M' \cap Q} y_{new}(e) \qquad (*) \\ &= y_{new}(u) + y_{new}(v) + w_i(M' \cap Q) \\ &\quad + w''_{new}(Q \cap M') \qquad \text{defn. of } w''_{new} \end{aligned}$$

Therefore,

$$\begin{aligned} (2.1) \qquad w_i(M'') &= w_i(M') + y_{new}(u) \\ &\quad + y_{new}(v) + w''_{new}(Q \cap M') \end{aligned}$$

(*) holds because beside u and v, the other possible difference of vertices in $Q \setminus M'$ and $Q \cap M'$ are those with zero y-values, which are the endpoints of P_u and P_v when $Q = P_u \cup P_v$.

Similarly, before the searches, we have:

$$\begin{aligned} (2.2) \qquad w_i(M) &= w_i(M') + y_{old}(u) \\ &\quad + y_{old}(v) - w''_{old}(P \setminus M') \end{aligned}$$

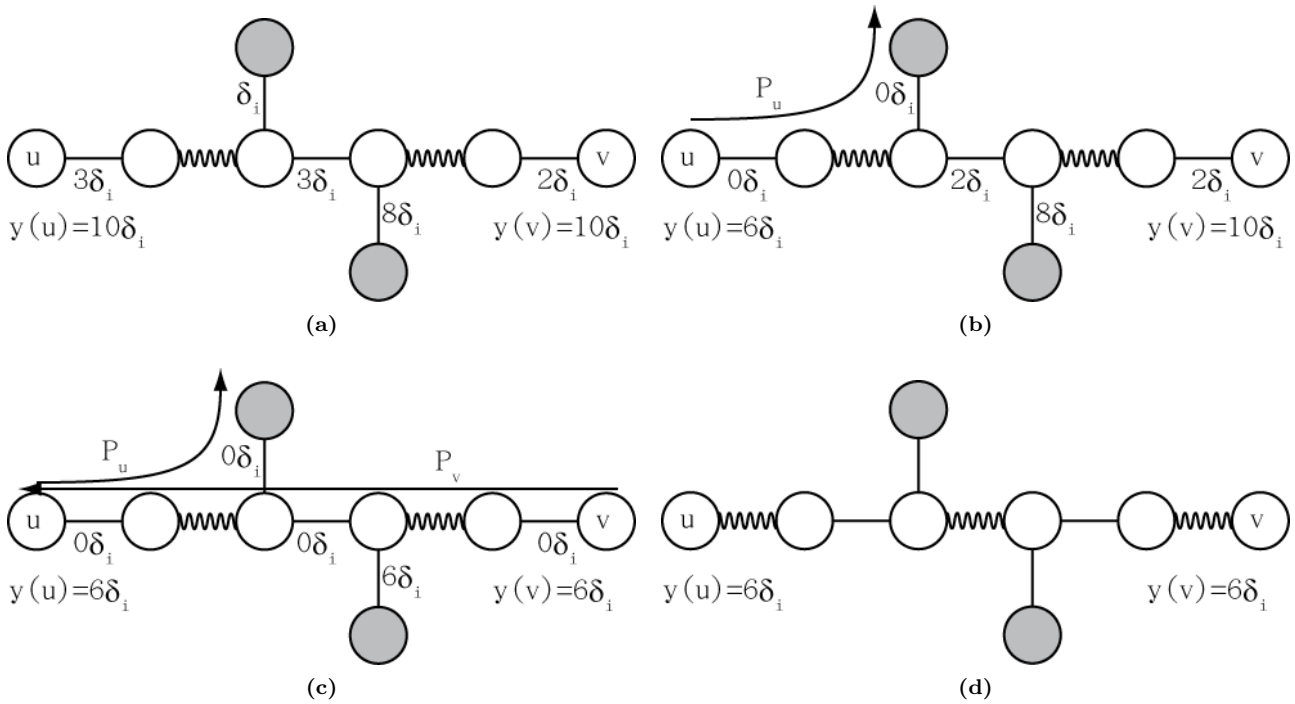

Figure 4: An example illustrating procedures for the chain case. Edges are shown with their new weight w'. The shaded vertices are free vertices with zero y-values. (a) After augmenting along P, u and v became free while having positive y-values. (b) $search(u)$ adjusted $\Delta_u = 4\delta_i$ and found an augmenting path P_u. (c) $search(v)$ adjusted $\Delta_v = 4\delta_i$ and found P_v. (d) Augmentation along Q. This is the case where there exists an augmenting path Q between u and v in $G[0,3]$, which happens to be P_v in the example.

The amount of dual adjustments is at most the distance between u and v, so $\Delta_u + \Delta_v \leq w''_{old}(P \setminus M') \leq 3n\delta_i$. Moreover:

$$\begin{aligned} w_i(M'') &\geq w_i(M') + y_{new}(u) + y_{new}(v) \\ &\qquad \text{by (2.1) and } w''_{new}(Q \cap M') \geq 0 \\ &= w_i(M') + y_{old}(u) + y_{old}(v) - \Delta_u - \Delta_v \\ &\geq w_i(M') + y_{old}(u) + y_{old}(v) - w''_{old}(P \setminus M') \\ &= w_i(M) \qquad \text{by (2.2)} \end{aligned}$$

LEMMA 2.8. *At most $O(\sqrt{n})$ rounds of augmentation and dual adjustment are required to reduce $f(B)$ to 0.*

Proof. When $f(B) = b$, choose $t = \sqrt{b}/2$. Either we can obtain an anti-chain B' of size at least $\lceil\sqrt{b}\rceil$ and decrease $f(B)$ by $\lceil\sqrt{b}/2\rceil$, or we can obtain a chain B' such that $f(B') \geq \lceil\sqrt{b}/2\rceil$ and decrease $f(B)$ by $f(B')$. In any case, we can decrease $f(B)$ by $\lceil\sqrt{b}/2\rceil$. The number of rounds is at most $T(b)$, where $T(b) = T(b - \lceil\sqrt{b}/2\rceil) + 1$ for $b > 0$ and $T(b) = 0$ for $b = 0$. It can be shown by induction that $T(b) \leq 4\sqrt{b}$, so that $T(b) \leq 4\sqrt{b} \leq 4\sqrt{2n}$.

2.3 Phase III The procedure for Phase III is similar to that for Phase II, but with several differences. First, instead of operating on $G[1,3]$, we will operate on $G[0,1]$ in this phase. Second, in the augmentation step, the definition of augmenting walks is modified such that the walk must contain at least one matched edge that is not tight. One exception is that an augmenting path in $G[0,3]$ of the chain case still refers to the old definition in Phase II, where we do not require it to contain at least one non-tight edge. Third, the way we find augmenting walks will be different from Phase II, since a tight edge in an augmenting walk will not become ineligible after an augmentation.

2.3.1 Phase III - Augmentation

LEMMA 2.9. *Each augmentation along the augmenting walk in $G[0,1]$ increases the weight of M. Consequently, there can be at most $\sqrt{n}$ augmenting walks in Phase III.*

Proof. Let M be the original matching and M' be the matching after augmentation. Suppose P is an augmenting walk. We must have $\sum_{v \in M \cap P} y(v) = \sum_{v \in M' \cap P} y(v)$, regardless of whether P is an augmenting cycle or an augmenting path. Since P contains at least one non-tight matched edge, $w(M \cap P) < \sum_{v \in M \cap P} y(v) = \sum_{v \in M' \cap P} y(v) = w(M' \cap P)$. Since

all weights are integers, the weight of the matching is increased by at least one.

After Phase II, $\delta_L = 2^{\lfloor \log N/\sqrt{n} \rfloor - \lceil \log N \rceil} \leq 1/\sqrt{n}$. By Lemma 2.1, we have $w(M) \geq w(M^*) - n\delta_L \geq w(M^*) - \sqrt{n}$. Since each augmentation increases the weight of M by at least one, and by Lemma 2.7, the weight of M does not decrease in dual adjustment steps, there can be at most $\sqrt{n}$ augmentations.

We have to ensure that no augmenting walks exist in $G[0,1]$ after the augmentation step. Since an augmenting walk may contain tight matched edges in $G[0,1]$, Lemma 2.2 and Lemma 2.3 no longer guarantee augmenting along a maximal set of augmenting cycles/paths will break up all eligible cycles/paths. However, by Lemma 2.9, we only need to find one augmenting walk in time $O(m)$ if it exists.

This can be done by the following procedure. First, obtain $\overrightarrow{G}[0,1]$ by orienting the edges of $G[0,1]$. If there is an augmenting cycle, then the cycle must contain a non-tight edge, say e. Also, the endpoints of e must be strongly connected in $\overrightarrow{G}[0,1]$. Therefore, to detect such cycles, run a strongly connected component algorithm first and then check whether the endpoints of non-tight edges are strongly connected.

Second, to detect an augmenting path, run the algorithm in $O(m)$ time described in Lemma 2.5 to determine whether v is adjustable for all $v \in G$. If there exists a non-tight matched edge uv such that both u and v are not adjustable, then there must be an augmenting path containing uv. Therefore, an augmenting walk can be found in $O(m)$ time, if one exists, and the total time spent on augmentation during Phase III is $O(m\sqrt{n})$.

2.3.2 Phase III - Dual Adjustment In Phase III, our goal is to tighten all non-tight edges. Thus, these edges are considered to be violated. That is, $B = \{e \in M : y(e) - w_i(e) = \delta_i\} = G[1,1] \cap M$. The definition of badness, $f(e)$, is changed to $(y(e) - w_i(e))/\delta_i$ accordingly. In Lemma 2.5, if u and v are both unadjustable, it is true that there will be an augmenting path containing a non-tight edge, which is uv. This will contradict with the fact that there are no augmenting paths after the augmentation step. Therefore, the lemma still works.

The other difference is Lemma 2.4, where the graph $\overrightarrow{G}$ may contain zero-length cycles or zero-length paths now. However, that does not affect how we select a chain or an anti-chain. The difference is that the graph may not be a DAG now but we still need to compute the length of the longest path from S to every vertex in linear time, where S is the set of vertices with zero in-degrees. This can be done by the following procedure:

1. Find the strongly connected components of $\overrightarrow{G}[0,0]$.
2. For each strongly connected component C in $\overrightarrow{G}[0,0]$, contract C into one vertex in $\overrightarrow{G}[0,1]$, because all vertices in C are supposed to have the same length of longest path from S in $\overrightarrow{G}[0,1]$.
3. Compute the longest path in the new contracted graph, which is a DAG.

Lemma 2.6, Lemma 2.7, and Lemma 2.8 all hold if we replace $G[1,3]$ by $G[0,1]$, and the existence of tight augmenting cycles/paths does not affect their correctness. Thus, the total time spent on dual adjustment in Phase III for $f(B)$ to reach zero is still $O(m\sqrt{n})$.

2.4 Maximum Weighted Perfect Matching Suppose a perfect matching exists in G. By the reduction described in Section 1, where we added nN weight to every edge, we can solve the MWPM problem in $O(m\sqrt{n}\log(nN))$ time. This does not improve the previous bound in [15]. However, below we give an algorithm that uses fewer scales, $\lceil \log(\sqrt{n}N) \rceil$ instead of $\lceil \log(nN) \rceil$. This is done by modifying the algorithm for MWM, where we maintain the following:

PROPERTY 2.2. *Let $\delta_0 = 2^{\lfloor \log N \rfloor}$, $L = \lceil \log(\sqrt{n}N) \rceil$. At the end of each scale $i \in [0, L]$, we maintain a perfect matching M with the following:*

1. ***(Granularity of** y**)** $y(u)$ *is a multiple of* δ_i.*
2. ***(Domination)*** $y(e) \geq w_i(e)$ *for all* $e \in E$.*
3. ***(Near Tightness)*** $y(e) \leq w_i(e) + 3\delta_i$. *At the end of scale i, it is tightened so that* $y(e) \leq w_i(e) + \delta_i$,*

For scale $i = 0$, find a perfect matching M using the Hopcroft-Karp algorithm in $O(m\sqrt{n})$ time [20], and assign $y(u) \leftarrow \delta_0$ to all left vertices u, $y(v) \leftarrow 0$ to all right vertices v.

Next, begin Phase II for scale $i \in [1, L]$ and Phase III at the end of scale L with the following modifications:

1. An augmenting walk only refers to an alternating cycle. We no longer consider augmenting paths so that we always keep the matching M to be perfect. More precisely, in the augmentation step, we no longer run *path_search*(x). In the dual adjustment step, if it is the anti-chain case, then either side of an edge in B' is adjustable, since now we allow the dual variables to have negative values. Therefore, for an anti-chain B' we can always decrease $f(B)$ by $|B'|$.

   In the chain case, the endpoints of P, say u and v, are freed temporarily. Here, we must force the

search(x) to find an augmenting path to connect u and v back. This can be done by only doing *search*(u) until the augmenting path in $G[0,3]$ between u and v opens up (i.e. force z_{min} to be v), which is always possible since we no longer need to keep y-values non-negative.

2. When $f(B) = b$, choose $t = \sqrt{b/2}$. Either we can obtain an anti-chain B' of size at least $\lceil\sqrt{b/2}\rceil$ and decrease $f(B)$ by $\lceil\sqrt{b/2}\rceil$, or we can obtain a chain B' such that $f(B') \geq \lceil\sqrt{b/2}\rceil$ and decrease $f(B)$ by $f(B')$. In any case, we can decrease $f(B)$ by $\lceil\sqrt{b/2}\rceil$, so the number of rounds is at most $T(b) = T(b - \lceil\sqrt{b}/2\rceil) + 1$. It can be shown by induction, $T(b) \leq 2\sqrt{2b} \leq 4\sqrt{n}$.

3. When Phase II ends at scale L, the result of Lemma 2.9 also holds. Since $\delta_L = 2^{\lfloor \log N\rfloor - \lceil\log(\sqrt{n}N)\rceil} \leq 1/\sqrt{n}$, by Lemma 2.1, $w(M) \geq w(M^*) - n\delta_L \geq w(M^*) - \sqrt{n}$. Thus, the matching can be improved at most $\sqrt{n}$ times.

Therefore, the algorithm runs in $\lceil\log(\sqrt{n}N)\rceil$ scales, where each scale takes $O(m\sqrt{n})$ time. The reason why we cannot achieve the same bound as the MWM problem is because Phase I does not apply. It is still unknown whether the MWPM problem can be solved in $O(m\sqrt{n}\log N)$ time.

3 Discussion

We believe that finding the MWM is easier than finding the MWPM. In [16], Gabow and Tarjan gave a scaling algorithm that solves the MWPM problem on general graphs in $O(m\sqrt{n\log n\alpha(n)}\log(nN))$ time. It is an interesting open problem whether the MWM on general graphs can be found in a faster time.

There are some reasons why it seems not possible to extend this algorithm directly to the general graph case, where there are blossoms. First, after entering the next scale, the edges in blossoms might not be tight or near tight anymore. It is likely one has to dissolve all these old blossoms. In [16], they throw away the old matching and try to find a new one while dissolving the old blossoms in every scale. Let C and D be two old blossoms such that $D \subseteq C$, $C \setminus D$ is called a *shell*. Their algorithm treats each shell independently until either C or D dissolves. However, it seems not possible to treat each shell independently in our algorithm, because we keep the old matching from the last scale and there might be matched edges crossing the shells. On the other hand, if we do not treat each shell independently, it will be unclear how we reduce $f(B)$ by $\sqrt{f(B)}$ in a round, because the chain/anti-chain argument does not seem to work when there are nested blossoms.

Second, even if there are no old blossoms in the previous scale, when we find an augmenting walk passing a blossom node, the edges inside the blossom become ineligible. This no longer guarantees that augmenting along the next augmenting walk passing this blossom will increase the weight of the matching, which makes Lemma 2.9 inapplicable.

Acknowledgements We would like to thank Seth Pettie for many useful comments, in particular, suggesting [1] to us.

References

[1] M. L. Balinski and R. E. Gomory. A primal method for the assignment and transportation problems. *Management Sci.*, 10(3):578–593, 1964.

[2] J. Cheriyan and K. Mehlhorn. Algorithms for dense graphs and networks on the random access computer. *Algorithmica*, 15(6):521–549, 1996.

[3] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[4] R. B. Dial. Algorithm 360: Shortest-path forest with topological ordering. *Comm. ACM*, 12:632–633, 1969.

[5] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Ann. Math.*, 51(1):161–166, 1950.

[6] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[7] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[8] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51(2):261–272, 1995.

[9] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[10] H. N. Gabow. An efficient implementation of Edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23(2):221–234, 1976.

[11] H. N. Gabow. Scaling algorithms for network problems. In *Proc. 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–257, 1983.

[12] H. N. Gabow. A scaling algorithm for weighted matching on general graphs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 90–100, 1985.

[13] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 434–443, 1990.

[14] H. N. Gabow, Z. Galil, and T. H. Spencer. Efficient implementation of graph algorithms using contraction. *J. ACM*, 36(3):540–572, 1989.

[15] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[16] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991.

[17] Z. Galil, S. Micali, and H. N. Gabow. An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM J. Comput.*, 15(1):120–130, 1986.

[18] A. V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24(3):494–504, 1995.

[19] N. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM J. Comput.*, 39(2):679–702, 2009.

[20] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[21] C. G. J. Jacobi. De investigando ordine systematis aequationum differentialum vulgarium cujuscunque. *Borchardt Journal für die reine und angewandte Mathematik*, LXIV(4):297–320, 1865. Reproduced in *C. G. J. Jacobi's gesammelte Werke, fünfter Band*, K. Weierstrass, Ed., Berlin, Bruck und Verlag von Georg Reimer, 1890, pp. 193–216.

[22] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.

[23] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM J. Comput.*, 31(1):18–26, 2001.

[24] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[25] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *Proc. 45th Symp. on Foundations of Computer Science (FOCS)*, pages 248–255, 2004.

[26] J. Munkres. Algorithms for the assignment and transportation problems. *J. Soc. Indust. Appl. Math.*, 5:32–38, 1957.

[27] F. Ollivier. Looking for the order of a system of arbitrary ordinary differential equations. *Appl. Algebra Eng. Commun. Comput.*, 20(1):7–32, 2009. English translation of: C. G. J. Jacobi, De investigando ordine systematis aequationum differentialum vulgarium cujuscunque," *Borchardt Journal für die reine und angewandte Mathematik* 65(4), 1865, pp. 297–320, also reproduced in: C. G. J. Jacobi, *Gesammelte Werke, Vol. 5, K. Weierstrass, Ed., Berlin, Bruck und Verlag von Georg Reimer, 1890, pp. 193–216.*

[28] J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Math. Program.*, 54:41–56, 1992.

[29] P. Sankowski. Weighted bipartite matching in matrix multiplication time. In *Proceedings 33rd Int'l Symposium on Automata, Languages, and Programming (ICALP)*, pages 274–285, 2006.

[30] M. Thorup. Equivalence between priority queues and sorting. In *Proc. 43rd Symp. on Foundations of Computer Science (FOCS)*, pages 125–134, 2002.

[31] M. Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *J. Comput. Syst. Sci.*, 69(3):330–353, 2004.

[32] N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.