Pearls of Algorithm

2. Bipartite Matching and the Fast Fourier Transform

Our goal is to learn efficient solutions of two important problems, the computation of a

- maximum (weighted) matching in bipartite graphs
- Fourier transform.

In practice, both problems occur often as a subproblem which has to be solved.

Moreover, we shall be concerned with fundamental methods of the development of algorithms as

- the augmenting solution method,
- the primal-dual method and
- the divide and conquer paradigm.

2.1 Bipartite Matching

References

C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall 1982.

N. Blum, Algorithmen und Datenstrukturen, Oldenbourg 2004 (in German).

2.1.1 Definitions and the general method

A graph $G = (V, E)$ consists of a finite, nonempty set of nodes V and a set of edges E. G is either directed or undirected. In the (un-)directed case, each edge is an (un-)ordered pair of distinct nodes. A graph $G = (V, E)$ is bipartite if V can be partitioned into disjoint nonempty sets A and B such that for all $(u, v) \in E$, $u \in A$ and $v \in B$, or vice versa. Then we often write $G = (A, B, E)$.

A path P from $v \in V$ to $w \in V$ is a sequence of nodes

$$P = v = v_0, v_1, \ldots, v_k = w$$

which satisfies $(v_i, v_{i+1}) \in E$, for $0 \leq i < k$. The length $|P|$ of P is the number k of edges on P.

P is simple if $v_i \neq v_j$, for $0 \leq i < j \leq k$.

If there exists a path from v to w (of length 1) v is called a (direct) predecessor of w, and w is called a (direct) successor of v.

Let $G = (V, E)$ be an undirected graph. $M \subseteq E$ is a matching of G if no two edges in M have a common node. A matching M is maximal if there exists no $e \in E \setminus M$ such that $M \cup \{e\}$ is a matching. A matching M is maximum if there exists no matching $M' \subseteq E$ of larger size.

Given an undirected graph $G = (V, E)$, the maximum matching problem is finding a maximum matching $M \subseteq E$.

A path $P = v_0, v_1, \ldots, v_k$ is M-alternating if it contains alternately edges in M and in $E \setminus M$. A node $v \in V$ is M-free if v is not incident to an edge in M.

Let $P = v_0, v_1, \ldots, v_k$ be a simple M-alternating path. P is M-augmenting if v_0 and v_k are M-free. Let P be an M-augmenting path in G. Then $M \oplus P$ denotes the symmetric difference of M and P; i.e.

$$M \oplus P = M \setminus P \cup P \setminus M.$$

Note that $M \oplus P$ is a matching of G, and $|M \oplus P| = |M| + 1$.

Theorem 2.1 (Berge 1957)
Let $G = (V, E)$ be an undirected graph and $M \subseteq E$ be a matching of G. Then M is maximum if and only if there exists no M-augmenting path in G.

Proof:

„⇒"

If M is maximum then no M-augmenting path P can exist. Otherwise, $M \oplus P$ would be a matching of size $|M| + 1$, a contradiction.

„⇐"

Let M be a matching of G. Assume that in G no M-augmenting path exists.

Let M' be any maximum matching of G. We have to prove $|M'| = |M'|$

Assume

$$|M| < |M'|.$$

Let us consider the subgraph

$$G' := (V, M \oplus M').$$

M and M' matchings $\Rightarrow$

$$\deg_{G'}(v) \leq 2 \quad \forall v \in V.$$

Furthermore

$$\deg_{G'}(v) = 2 \Rightarrow$$

One of the two edges with end node v is in M and the other such a edge is in M'.

Hence, the connected components of G' are of the following kinds.

i) isolated nodes,
ii) cycles of even length with alternately edges in M and in M', or
iii) paths with alternately edges in M and in M'.

$|M'| > |M| \Rightarrow$

At least one of the connected components is a path P with more edges in M' than in M.

$\Rightarrow$

The first edge of P and also the last edge of P are in M'.

$\Rightarrow$

Both end nodes of P are M-free.

$\Rightarrow$

P is a an M-augmenting path.

This is a contradiction to the assumption that no M-augmenting path in G exists. ∎

Berge's theorem implies the following general method for finding a maximum matching in a graph G.

Algorithm MAXIMUM MATCHING

Input: An undirected graph $G = (V, E)$, and a matching $M \subseteq E$ (possibly $M = \emptyset$)

Output: A maximum matching M_{max}

Method:

```
while there exists an M-augmenting path
    do
        construct such a path P;
        M := M ⊕ P
    od;
M_max := M.
```

The key problem is now this:

How to find an M-augmenting path P, if such a path exists?

Next we will define the maximum weighted matching problem.

Let $G = (V, E)$ be an undirected graph. If we associate with each edge $(i,j) \in E$ a weight $w_{ij} > 0$ then we obtain a weighted undirected graph $G = (V, E, w)$.

The weight $w(M)$ of a matching M is the sum of the weights of the edges in M. A matching $M \subseteq E$ has <u>maximum weight</u> if

$$\sum_{(i,j) \in M'} w_{ij} \leq \sum_{(i,j) \in M} w_{ij} \quad \forall \text{ matchings } M' \subseteq E.$$

Given a weighted undirected graph $G = (V, E, w)$, the <u>maximum weighted matching problem</u> is finding a matching $M \subseteq E$ of maximum weight.

2.1.1 <u>The unweighted case</u>

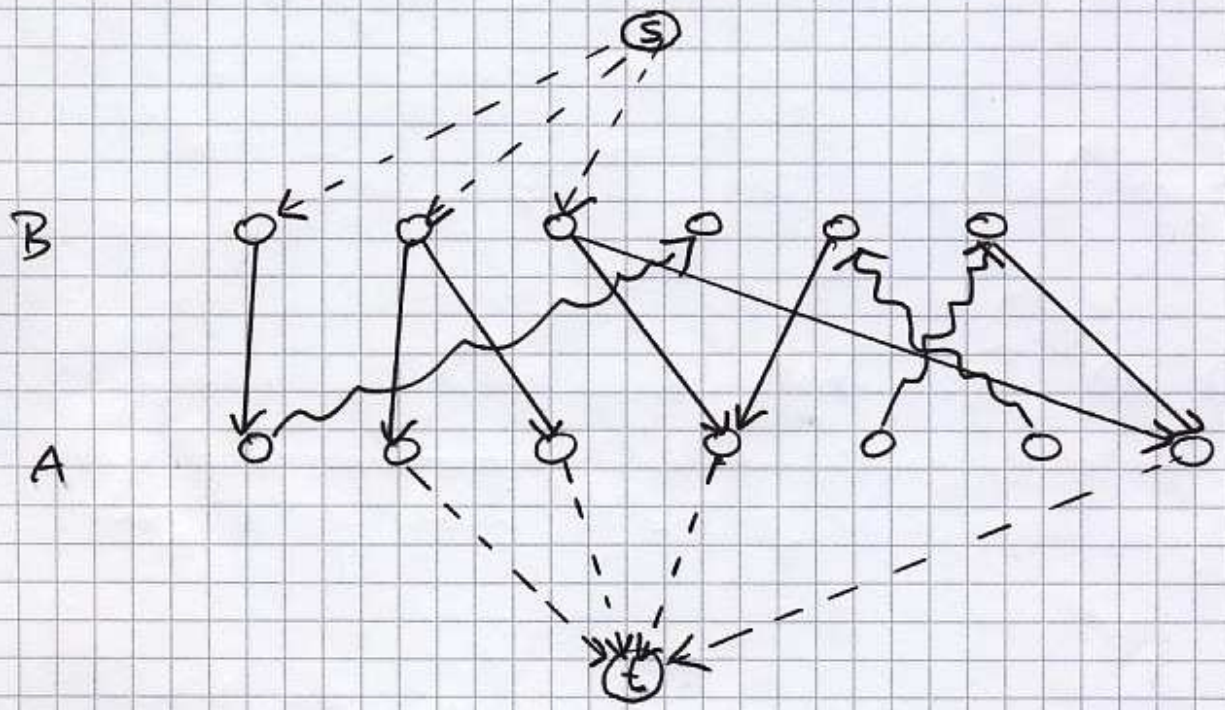
We solve the key problem above in the following way.

1) We reduce the key problem to a reachability problem in a directed, bipartite graph $G_M = (A', B', E_M)$.

2) We solve this reachability problem constructively.

Given the bipartite graph $G = (A, B, E)$ and the matching $M \subseteq E$, we direct the edges in M from A to B and the edges in $E \setminus M$ from B to A. Additionally, we add two new nodes s and t to $A \cup B$, add for each M-free node $b \in B$ the edge (s, b) to E_M, and add for each M-free node $a \in A$ the edge (a, t) to E_M

$\rightsquigarrow$



$\rightsquigarrow$ edge in M

$\longrightarrow$ edge in $E \setminus M$

$\dashrightarrow$ additional edge

$G_M = (A', B', E_M)$ where

$A' = A \cup \{s\}$, $B' = B \cup \{t\}$, $s, t \notin A \cup B$, $s \neq t$

and

$$E_M = \{(u,v) \mid (u,v) \in M \text{ and } u \in A, v \in B\}$$
$$\cup \{(x,y) \mid (x,y) \in E \setminus M \text{ and } x \in B, y \in A\}$$
$$\cup \{(s,b) \mid b \in B \text{ } M\text{-free}\}$$
$$\cup \{(a,t) \mid a \in A \text{ } M\text{-free}\}.$$

Lemma 2.1

Let $G = (A, B, E)$ be a bipartite graph, $M \subseteq E$ be a matching and $G_M = (A', B', E_M)$ constructed as above. Then there exists an M-augmenting path in G if and only if there is a simple path from s to t in G_M.

Proof:

„⇒"

Let $P = v_0, v_1, v_2, \ldots, v_k$ be an M-augmenting path in G.

W.l.o.g. we can assume $v_0 \in B$.

⇒

1) $v_i \in A$ for i odd,
2) $v_i \in B$ for i even,
3) $v_0 \in B$ M-free and $v_k \in A$ M-free,
4) $(v_i, v_{i+1}) \in M$ if i odd, and
5) $(v_i, v_{i+1}) \in E \setminus M$ if i even.

⇒

By the construction of G_M, the path

$$P' := s, v_0, v_1, v_2, \ldots, v_k, t$$

is a simple path from s to t in G_M.

„$\Leftarrow$" analogously ∎

The following algorithm constructs an M-augmenting path if such a path exists.

Algorithm FIND AUG PATH

Input: A bipartite graph $G = (A, B, E)$ and a matching $M \subseteq E$.

Output: An M-augmenting path P if such a path exists

Method:

(1) Construct $G_M = (A', B', E_M)$.
(2) Apply a depth first search with start node s to G_M.
(3) If depth first search reaches the node t then put out the path P contained in the stack without the nodes s and t.

Next we shall analyze the time used by the algorithm FIND AUG PATH.

Let
$$n = |A| + |B| \quad \text{and} \quad m = |E|.$$

Obviously, G_M can be constructed in $O(n+m)$ time. Depth first search uses also only $O(n+m)$ time. Hence, the algorithm FINDAUGPATH constructs an M-augmenting path in $O(n+m)$ time if such a path exists.

The symmetric difference $M := M \oplus P$ can be performed in $O(|P|) = O(n)$ time. At most $\lceil \frac{n}{2} \rceil$ augmentations can be performed. Altogether, we have proven the following theorem.

Theorem 2.2.
Let $G = (A, B, E)$ be a bipartite graph which contains no isolated nodes. Then we can compute a maximum matching of G in $O(n \cdot m)$ time where $n = |A| + |B|$ and $m = |E|$.

In general graphs $G = (V, E)$, the solution of the key problem above is much more complicated.

2.1.2 The weighted case

Let $G = (V, E, w)$ be a weighted undirected graph and $M \subseteq E$ be a matching of G. Let P be a simple M-alternating path.

The gain $\Delta(P)$ with respect to P denotes the weight change of the matching after the performance of the symmetric difference $M := M \oplus P_j$ i.e.,

$$\Delta(P) = \sum_{(i,j) \in P \wedge E \setminus M} w_{ij} - \sum_{(i,j) \in P \wedge M} w_{ij}.$$

The following theorem gives an exact characterization of a maximum weighted matching of a weighted undirected graph.

Theorem 2.3

Let $G = (V, E, w)$ be a weighted undirected graph and $M \subseteq E$ be a matching of G. Then M is of maximum weight if and only if none of the following two cases is fulfilled:

1) There is an M-alternating cycle P of even length with $\Delta(P) > 0$.
2) There is a simple M-alternating path P such that
   i) $(v,w) \in M \Rightarrow (v,w) \in P$ or $v, w \notin P$ and
   ii) $\Delta(P) > 0$.

Proof:

„$\Rightarrow$"

Let $M \subseteq E$ be a matching of maximum weight. Obviously, none of the both cases can be

fulfilled. Otherwise, after $M := M \oplus P$ we would obtain a matching of larger weight.

"$\Leftarrow$"

Let $M' \subseteq E$ be any matching of G. Consider the graph

$$G' := (V, M \oplus M', w).$$

Each connected component of G' which is not a isolated node is either an M-alternating cycle P of even length or a simple M-alternating path P. Hence, for each connected component P we have

$$\Delta(P) \leq 0.$$

$\Rightarrow$

$$w(M') = w(M \oplus M' \oplus M) \leq w(M).$$

■

Applying Theorem 2.3, we have to consider M-augmenting paths and also the other simple M-alternating paths and als the M-alternating cycles. The following theorem gives us the possibility to restrict us to the consideration of M-augmenting paths.

Theorem 2.4

Let $G = (V, E, w)$ be a weighted undirected graph and $M \subseteq E$ be a matching of G such that $|M| = k$ and $w(M)$ is maximum with respect to

matchings of size k. Let P be an M-augmenting path with maximal gain $\Delta(P)$. Then $M' := M \oplus P$ is a matching with maximum weight within the matchings of size $k+1$ of G.

Proof:

Let M'' be any matching of G with $|M''| = k+1$. It suffices to prove

$$w(M'') \leq w(M \oplus P).$$

Consider the subgraph

$$G' := (V, M'' \oplus M, w).$$

Let R be any M-augmenting path in G'. Note that R exists since $|M''| > |M|$.

$\Rightarrow$

$$w(M) + \Delta(R) = w(M \oplus R) \leq w(M \oplus P)$$
$$w(M'') - \Delta(R) = w(M'' \oplus R) \leq w(M)$$

We obtain after the addition of both inequalities

$$w(M) + w(M'') \leq w(M \oplus P) + w(M)$$

$\Leftrightarrow$

$$w(M'') \leq w(M \oplus P)$$

■

Applying Theorem 2.3 and Theorem 2.4 we obtain the following general method for

the computation of a maximum weighted matching of a graph $G = (V, E, w)$.

Algorithm MAXWEIMATCHING

Input: A weighted undirected graph $G = (V, E, w)$

Output: A maximum weighted matching M_{max} of G.

Method.

$M := \emptyset$;
while there exists an M-augmenting path P with $\Delta(P) > 0$
do
construct such a path with maximal gain $\Delta(P)$;
$M := M \oplus P$
od ;
$M_{max} := M$.

The following theorem proves the correctness of the algorithm above.

Theorem 2.5
The algorithm MAXWEIMATCHING a matching of maximum weight of G.

Proof:

Let $|M_{max}| = t$. Then by Theorem 2.4

- For $1 \leq k \leq t$, the matching M_k computed during the kth performance of the body of the while-loop has maximum weight within all matchings of size k of G.

Assume that M_{max} has not maximum weight.

Theorem 2.3 $\Rightarrow$

One of the following cases is fulfilled:

1) $\exists$ M_{max}-alternating cycle P of even length with $\Delta(P) > 0$.
2) $\exists$ simple M_{max}-alternating path P such that
   i) $(v,w) \in M_{max} \Rightarrow (v,w) \in P$ or $v,w \notin P$ and
   ii) $\Delta(P) > 0$.

If there is an M_{max}-alternating cycle P of even with $\Delta(P) > 0$ then $M_{max} \oplus P$ would be a matching of size t with

$$w(M_{max} \oplus P) > w(M_{max}).$$

This contradicts that M_{max} is a matching of size t of maximum weight within all matchings of size t.

If there is an M_{max}-alternating path P such that

i) $(v,w) \in M_{max} \Rightarrow (v,w) \in P$ or $v, w \notin P$ and
ii) $\Delta(P) > 0$

then P cannot be M_{max}-augmenting. Otherwise, the algorithm would not terminate with the matching M_{max}.

If the length of P is even then $M_{max} \oplus P$ would be a matching of size t with

$$w(M_{max} \oplus P) > w(M_{max}),$$

a contradiction.

If the length of P is odd then $M_{max} \oplus P$ would be a matching of size t-1 with

$$w(M_{max} \oplus P) > w(M_{t-1}),$$

a contradiction.

Altogether, Theorem 2.3 implies that the computed matching M_{max} is of maximum weight. ∎

The key problem is now this:

How to find an M-augmenting path P with maximum gain $\Delta(P) > 0$ if such a path exists?

Note that the considerations above holds also for nonbipartite graphs.

Goal:
Development of a solution of the key problem for bipartite graphs.

Let $G = (A, B, E, w)$ be a weighted undirected graph and $M \subseteq E$ be a matching of G.

Analogous to the unweighted case, we construct the weighted directed graph

$$G_M = (A \cup \{s\}, B \cup \{t\}, E_M, w)$$

where the new edges (s,b) and (a,t) obtains the weight 0.

Observation:
A depth first search on G_M with start node s finds an M-augmenting path P if such a path exists. But $\Delta(P)$ must not be maximal within all M-augmenting paths.

$\Rightarrow$

We need a mechanism which guarantees that $\Delta(P)$ is maximal within all M-augmenting paths.

$\rightsquigarrow$

The primal-dual method for the weighted bipartite matching problem.

The primal-dual method can be separated into rounds. Each round decides into two steps, the <u>search step</u> and the <u>extension step</u>.

The input of a search step will always be a subgraph G_M^* of G_M and an upper bound U such that

1) $\Delta(P) \leq U$ for all M-augmenting paths P in G_M,
2) $\Delta(P) = U$ for all M-augmenting paths P in G_M^*, and
3) G_M^* contains all M-augmenting paths P of G_M with $\Delta(P) = U$.

For the construction of an M-augmenting path in G_M^* if such a path exists the search step will use depth first search. If G_M^* does not contain any M-augmenting path, the extension step will compute the input for the next search step.

Given G_M^*, the search step will be performed in the same way as in the unweighted case. It remains the description of the extension step.

Given $G = (A, B, E, w)$ we construct the graph

$$G_\phi = (A \cup \{s\}, B \cup \{t\}, E_\phi, w).$$

Let

$$W := \max \{ w_{ij} \mid (i,j) \in E \}.$$

We initialize the upper bound U and the input graph $G^*_\emptyset$ for the first search step as follows:

$$U := W \quad \text{and}$$
$$G^*_\emptyset := (A \cup \{s\}, B \cup \{t\}, E^*_\emptyset, w)$$

where

$$E^*_\emptyset = \{(i,j) \mid (i,j) \in E_\emptyset \text{ and } w_{ij} = W\} \cup \{(s,i) \mid i \in B\} \cup \{(j,t) \mid j \in A\}.$$

It is easy to prove that U and $G^*_\emptyset$ fulfill the Properties 1 - 3.

Exercise:
Prove that U and $G^*_\emptyset$ fulfill the properties 1 - 3.

Assume that the current search step terminates with

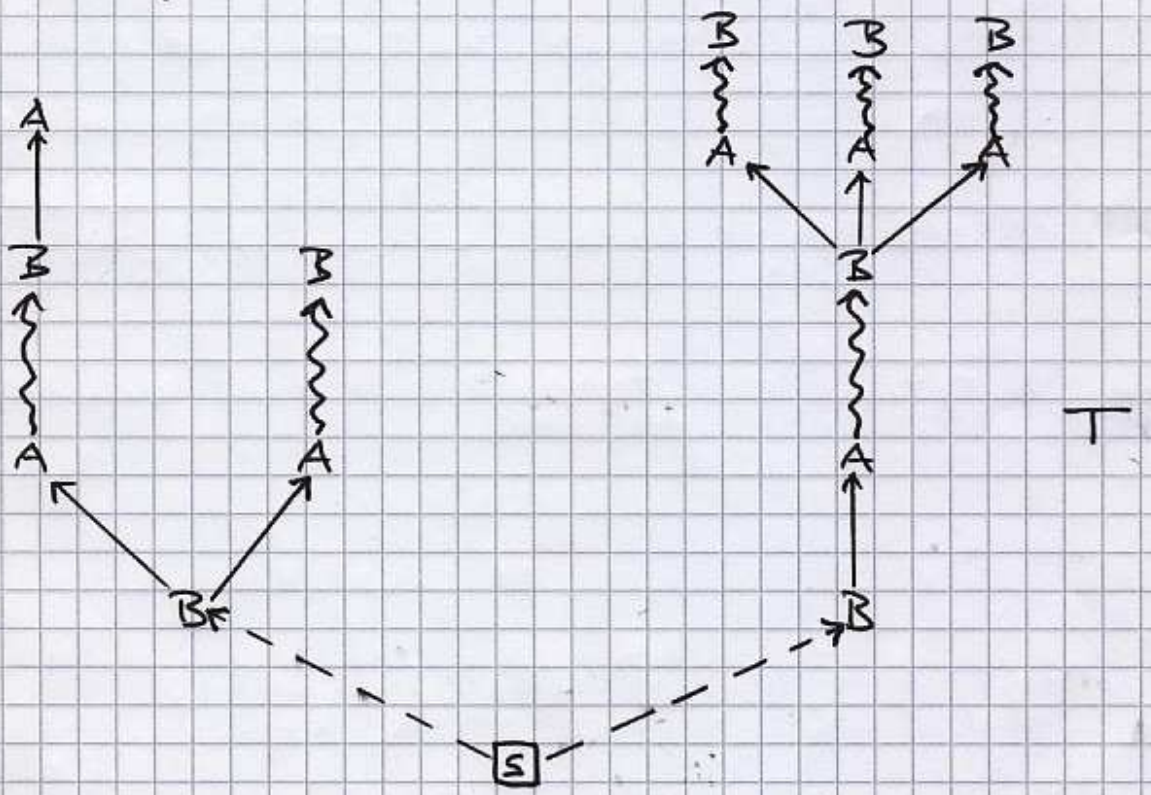
- a matching M,
- a weighted directed graph
  $$G_M = (A \cup \{s\}, B \cup \{t\}, E_M, w)$$
  and
- the input graph
  $$G^*_M = (A \cup \{s\}, B \cup \{t\}, E^*_M, w)$$
  of the last unsuccessful depth first search.

Note that every M-augmenting path P in G_M has gain $\Delta(P) < U$.

Question:

How to get the reduced upper bound and the input graph for the next search step?

For getting an answer of this question let us consider the depth first search tree T which has been constructed during the last unsuccesful depth first search.



Let

$$B_T := B \cap T \quad \text{and} \quad A_T := A \cap T.$$

Goal:

The reduction of the current upper bound U by the appropriate value δ and the extension of E_M^* by edges from $E \setminus E_M^*$ such that the

Properties 1-3 will be fulfilled with respect to the new upper bound $u-\delta$ and the constructed input graph for the next search step.

Question:
How to get the appropriate δ and in dependence of δ those edges which we have to add to E_M^*?

Idea
We associate with all nodes of the graph node weights such that the gain of a path P can be computed by the only consideration of the end nodes of the path P.

More exactly, we associate with each node $j \in A$ a node weight v_j and with each node $i \in B$ a node weight u_i such that always the following invariants are fulfilled:

1) $u_i + v_j \geq w_{ij}$ for all $(i,j) \in E \setminus M$,
2) $v_j + u_i = w_{ji}$ for all $(j,i) \in M$,
3) $u_i = u$ for all M-free $i \in B$, and
4) $v_j = 0$ for all M-free $j \in A$.

Then we define

$$E_M^* := M \cup \{(i,j) \in E_M \setminus M \mid u_i + v_j = w_{ij}\}.$$

For the computation of the appropriate value δ we define for $i \in B$ and $j \in A$ unbalances

π_{ij} and π_{ji} in the following way.

$$\pi_{ij} := u_i + v_j - w_{ij} \quad \text{and}$$

$$\pi_{ji} := v_j + u_i - w_{ji}.$$

Goal:
Prolongation of paths in the depth first search tree T by adding appropriate edge (i,j) with $i \in B_T$ and $j \in A \setminus A_T$. The upper bound u should be reduced as few as possible.

Note that $(i,j) \notin E_M^*$.

$\Rightarrow$

$u_i + v_j > w_{ij} \Rightarrow \pi_{ij} > 0$.

Question:
How to change the node weights u_i an v_j such that the edge (i,j) is added to E_M^*; i.e., $\pi_{ij} = 0$?

After the reduction of u_i by $\delta := \pi_{ij}$ we obtain $(u_i - \pi_{ij}) + v_j = 0$ such that the edge (i,j) would be added to E_M^*. Since we do not wish to delete some edge from E_M^*, we have to modify the node weights of the other nodes in T in an appropriate manner.

We perform

$$u_i := u_i - \delta \quad \forall\, i \in B_T \quad \text{and}$$
$$v_j := v_j + \delta \quad \forall\, j \in A_T$$

Furthermore, we reduce the upper bound.

$$u := u - \delta.$$

Properties:

- At the beginning

$$u_i = u \quad \forall \text{ M-free nodes } i \in B.$$

Always we have $i \in B_T$ $\forall$ M-free $i \in B$.

$\Rightarrow$

After the extension step, we have

$$u_i = u \quad \forall \text{ M-free nodes } i \in B.$$

- At the beginning

$$v_j = 0 \quad \forall \text{ M-free nodes } j \in A.$$

Always we have $j \notin A_T$ $\forall$ M-free $j \in A$

$\Rightarrow$

After the extension step, we have

$$v_j = 0 \quad \forall \text{ M-free nodes } j \in B.$$

Since δ should be chosen such that

i) at least one edge is added to E_M^* and
ii) δ should be chosen as small as possible,

we define

$$\delta := \min \{ \pi_{ij} \mid i \in B_T \text{ and } j \in A \setminus A_T \}.$$

If $\delta \geq u$ then the algorithm terminates.

Exercise
Work out the algorithm in detail.

Let P be a path in G_M. The unbalance $\pi(P)$ of the path P is defined as follows.

$$\pi(P) := \sum_{e \in P} \pi_e.$$

Next we will show that after the extension step with respect to the new upper bound and the new input graph for the next search step the invariants 1-3 are fulfilled.

Lemma 2.2
Let $P = v_0, v_1, \ldots, v_k$ with $v_0 \neq s$ and $v_k \neq t$ be a simple alternating path in G_M. Then

$$\Delta(P) = \begin{cases} u_{v_0} + v_{v_k} - \pi(P) & \text{if } v_0 \in B, v_k \in A \\ u_{v_0} - u_{v_k} - \pi(P) & \text{if } v_0, v_k \in B \\ -v_{v_0} - u_{v_k} - \pi(P) & \text{if } v_0 \in A, v_k \in B \\ -v_{v_0} + v_{v_k} - \pi(P) & \text{if } v_0, v_k \in A. \end{cases}$$

Proof:

Definition $\Rightarrow$

$$\Delta(P) = \sum_{e \in P \cap E_M \setminus M} w_e - \sum_{e \in P \cap M} w_e$$

$$= \sum_{(i,j) \in P \cap E_M \setminus M} (u_i + v_i - \pi_{ij})$$

$$- \sum_{(j,i) \in P \cap M} (v_j + u_i - \pi_{ji})$$

Since $\pi_{ji} = 0$ for $(j,i) \in M$ there holds

$$= \underbrace{\left(\sum_{(i,j) \in P \cap E_M \setminus M} (u_i + v_j) - \sum_{(j,i) \in P \cap M} (v_j + u_i) \right)}_{D} - \pi(P)$$

Since P is an alternating path, up to the first summand with respect to the first edge on P and the second summand with respect to the last edge on P all summands vanish. Hence

$$D = \begin{cases} u_{v_0} + v_{v_k} & \text{if } v_0 \in B, v_k \in A \\ u_{v_0} - u_{v_k} & \text{if } v_0, v_k \in B \\ -v_{v_0} - u_{v_k} & \text{if } v_0 \in A, v_k \in B \\ -v_{v_0} + v_{v_k} & \text{if } v_0, v_k \in A \end{cases}$$

This proves the lemma. ∎

Lemma 2.3

The algorithm fulfills always the following invariants:

i) $u_i = u$ for all M-free nodes $i \in B$,
ii) $v_j = 0$ for all M-free nodes $j \in A$, and
iii) $\pi_e \geq 0$ for all edges $e \in E_M$.

Proof:

Definition $\Rightarrow$

At the beginning, i.e., $M = \emptyset$ all invariants are fulfilled.

Goal:

The proof of

invariants fulfilled before the search and the extension step, respectively

$\Rightarrow$

the invariants are fulfilled after the search and the extension step, respectively.

The search step never changes the node weights or the upper bound. Hence, the invariants remains fulfilled after the search step.

We have shown above that the extension step maintains the invariants i) and ii).

δ is chosen in such a way such that the modification of the node weights never violate the invariant (ii).

■

Theorem 2.6

The algorithm MAXWEIGHTMATCHING implemented as described above terminates with a maximum weighted matching M_{max}.

Proof:

It is clear that the algorithm terminates with a matching M_{max}.

Theorem 2.4 $\Rightarrow$

It suffices to show that always an augmenting path P with maximal gain $\Delta(P)$ is augmented by the algorithm and that after the termination of the algorithm no augmenting path P with $\Delta(P) > 0$ exists.

Assume that the algorithm augment the M-augmenting path P with $\Delta(P) > 0$ although an M-augmenting path Q with $\Delta(Q) > \Delta(P)$ exists.

Let

$P = i, \ldots, j$ and $Q = p, \ldots, k$.

Lemma 2.2 $\Rightarrow$

$\Delta(Q) = u_p + v_k - \pi(Q)$ and
$\Delta(P) = u_i + v_j - \pi(P)$.

Lemma 2.3 $\Rightarrow$

$\pi(Q) \geq 0$, $u_i = u_p$ and $v_j = v_k = 0$.

Since $\pi(P) = 0$ we obtain

$$\Delta(Q) + \pi(Q) = \Delta(P)$$

and hence,

$$\Delta(Q) \leq \Delta(P)$$

a contradiction.

After the termination of the algorithm there hold

$u_i \leq 0$ for all M_{max}-free nodes $i \in B$ and
$v_j = 0$ for all M_{max}-free nodes $j \in A$.

Lemmas 2.2 and 2.3 $\Rightarrow$

$$\Delta(P) = u_i + v_j - \pi(P) \leq -\pi(P) \leq 0$$

for all M_{max}-augmenting paths $P = i, \dots, j$.

$\Rightarrow$

$\nexists$ M_{max}-augmenting path P with $\Delta(P) > 0$. ∎

Exercise:

Show that the algorithm MAXWEIMATCHING can be implemented such that its runtime is $O((|A|+|B|)^3)$.

2.2 The Fast Fourier Transform

2.2.1 Motivation

Let us consider the following problem.

- Given the "usal" representation of an input, the goal is to obtain from the input an output which fulfills some properties. The output should be represented in the same way as the input.

Example 2.1:

Given the coefficient representation of two polynomials

$$p(x) = \sum_{i=0}^{n} a_i x^i \quad \text{and} \quad q(x) = \sum_{i=0}^{n} b_i x^i$$

our goal is to get the coefficient representation of the product polynomial

$$p(x) \cdot q(x) = \sum_{k=0}^{2n} c_k x^k,$$

where

$$c_k = \sum_{i=\max\{0,\,k-n\}}^{k} a_i b_{k-i}.$$

◇

Example 2.2

Given a signal (i.e. speech) during a time interval where the signal is interspersed with noise, our goal is the removal of noise from the signal.

◊

Instead of to operate directly on the given representation of the input, it might be better

1. first to transform the given representation of the input to another representation,
2. to operate on the other representation of the input, and
3. to apply the inverse transformation to the output of Step 2 getting the output in the same representation as the input.

Example 2.1 (continuation)

Given the polynomials $p(x)$ and $q(x)$ by their coefficients

$(a_0, a_1, a_2, \ldots, a_n)$ and $(b_0, b_1, b_2, \ldots, b_n)$

we have to compute the vector of coefficients

$(c_0, c_1, c_2, \ldots, c_{2n})$

of the product polynomial $pq(x)$. This vector can be computed by performing

$$c_k := \sum_{i=\max\{0,\, k-n\}}^{k} a_i b_{k-i}$$

for $0 \leq k \leq 2n$. Then we would perform $\Omega(n^2)$ arithmetic operations leading to an $\Theta(n^2)$ algorithm for the multiplication of two polynomials of degree n.

Another unique representation of a polynomial of degree n is the so-called point-value representation.

A <u>point-value representation</u> of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ is a set of $n+1$ point-value pairs $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ such that

i) $x_i \neq x_j$ for $0 \leq i < j \leq n$ and
ii) $y_i = p(x_i)$ for $0 \leq i \leq n$.

A polynomial has many different point-value representations since any set of $n+1$ distinct points $x_0, x_1, \ldots, x_n$ can be used as a basis for the representation.

Given a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ in coefficient form, a point-value represen-

tation of $p(x)$ can be computed by

(1) Choose any set of $n+1$ distinct points $\alpha_0, \alpha_1, \ldots, \alpha_n$.

(2) For $0 \leq i \leq n$ evaluate $p(\alpha_i)$.

For the evaluation of $p(\alpha_i)$ we can use Horner's rule

$$p(x_i) = a_0 + \alpha_i(a_1 + \alpha_i(a_2 + \ldots + \alpha_i(a_{n-1} + \alpha_i a_n))\ldots).$$

Hence, the evaluation time for each α_i, $0 \leq i \leq n$ would be $\Theta(n)$ leading to a $\Theta(n^2)$ algorithm for the computation of a point-value representation from the coefficient representation of a polynomial of degree n.

The inverse of multipoint evaluation is determining the coefficient form of a polynomial from a point-value representation, the so-called interpolation.

Theorem 2.7 (Uniqueness of interpolation)

Let $\{(\alpha_0, y_0), (\alpha_1, y_1), \ldots, (\alpha_n, y_n)\}$ be any set of $n+1$ point-value pairs such that $\alpha_i \neq \alpha_j$ for $0 \leq i < j \leq n$. There is a unique polynomial $p(x)$ of degree n such that $y_i = p(\alpha_i)$ for $0 \leq i \leq n$.

Proof:

The following $n+1$ equations

$$y_i = p(\alpha_i), \quad 0 \le i \le n.$$

is equivalent to the matrix equation

$$\begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \dots & \alpha_0^n \\ 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \dots & \alpha_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

The $(n+1) \times (n+1)$ - matrix above is known as Vandermonde matrix and is denoted by

$$V(\alpha_0, \alpha_1, \dots, \alpha_n).$$

Exercise 2.1

Prove that the determinant of the Vandermonde matrix is

$$\det(V(\alpha_0, \alpha_1, \dots, \alpha_n)) = \prod_{0 \le j < k \le n} (\alpha_k - \alpha_j).$$

Hint:

Multiply column i by $-\alpha_0$ and add it to column $i+1$ for $i = n, n-1, \dots, 1$, and then use induction.

Since the Vandermonde matrix has determinant

$$\prod_{0 \le j < k \le n} (\alpha_k - \alpha_j)$$

the matrix is invertible if $\alpha_k \neq \alpha_j$, $0 \le j < k \le n$.

$\Rightarrow$

The matrix equation above has a unique solution and can be solved by

$$a = V(\alpha_0, \alpha_1, \ldots, \alpha_n)^{-1} \cdot y.$$

■

Note that the proof of Theorem 2.7 describes an algorithm for interpolation. Using Lagrange's formula, interpolation can be done in time $\Theta(n^2)$.

Since

$$pq(x) = p(x) \cdot q(x)$$

the point-value representation of polynomials is well-suited for multiplying polynomials. We can multiply two polynomials $p(x)$ and $q(x)$ of degree n in the following way

(1) Choose $2n+1$ distinct points $\alpha_0, \alpha_1, \ldots, \alpha_{2n}$ and compute by two multipoint evaluations $p(\alpha_0), p(\alpha_1), \ldots, p(\alpha_{2n})$ and $q(\alpha_0), q(\alpha_1), \ldots, q(\alpha_{2n})$

(2) Compute

$p(\alpha_0) \cdot q(\alpha_0),\ p(\alpha_1) \cdot q(\alpha_1),\ \ldots,\ p(\alpha_{2n}) \cdot q(\alpha_{2n})$

getting the point-value representation

$(\alpha_0, pq(\alpha_0)), (\alpha_1, pq(\alpha_1)), \ldots, (\alpha_{2n}, pq(\alpha_{2n}))$

(3) Compute the coefficient representation by interpolation.

The following mapping diagram illustrates this algorithm

$((a_0, a_1, \ldots, a_n), (b_0, b_1, \ldots, b_n))$ —polynomial multiplication→ $(c_0, c_1, \ldots, c_{2n})$

multipoint evaluation ↓ ; interpolation ↑

$((p(\alpha_0), \ldots, p(\alpha_{2n}), (q(\alpha_0), \ldots, q(\alpha_{2n}))$ —pointwise multiplication→ $(pq(\alpha_0), \ldots, pq(\alpha_{2n}))$

Although multipoint evaluation and interpolation can be done in $O(n^2)$ time such that the algorithm above would be an $O(n^2)$ algorithm, the constant is considerably larger than that of the school method.

Idea:
Choose the evaluation-interpolation points carefully such that multipoint evaluation and interpolation can be performed much faster than $O(n^2)$.

Fast Fourier Transformation.

◆

Example 2.2 (continuation)

A signal can be described either in the time domain, by the values of some quantity h as a function of time t, e.g., h(t), or else in the frequency domain, where the signal is specific by giving its amplitude H as a function of frequency f, that is H(f), with $-\infty < f < \infty$. Usually, we obtain a signal in the time domain.

Goal:
removal of the noise from the signal.

Since it is easier to remove noise from the representation of the signal in the frequency domain, we filter the noise from the signal in the following way

(1) Transform the representation of the signal in the time domain to the representation in the frequency domain.

(2) Remove the noise from the signal obtaining the representation of the resulting signal in the frequency domain.

(3) Transform the representation of the signal in the frequency domain to the representation in the time domain.

Both, the forward and the backward transformation can be done using the

Fast Fourier Transformation ◆

~~Integer multiplication is a widely used operation. Applying the school method we would obtain an $O(n^2)$ algorithm for the multiplication of two n-bit numbers.~~

~~Observation:~~
~~There are a lot of similarities between polynomial and integer multiplication.~~

~~⇝~~

~~Using the Fast Fourier Transform we obtain a much faster algorithm for the multiplication of two integers.~~

Contents:

2.2.2 The forward transform: Fast multipoint evaluation.

2.2.3 The inverse transform: Fast interpolation.

References:

A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley 1974.

A. Borodin, I. Munro, The Computational Complexity of Algebraic and Numeric Problems, U.M.I Out-of-Print Books on Demand 1990.

J.D. Lipson, Elements of algebra and algebraic computing, Addison-Wesley 1981.

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, 2nd Edu., Cambridge University Press 1992.

J. von zur Gathen, J. Gerhard, Modern Computer Algebra, Cambridge University Press 1999.

2.2.1 The forward transform: Fast multipoint evaluation

In the following, we assume that $n+1$ and m are powers of two; i.e., $n+1 = 2^r$ and $m = 2^s$ for some $r, s \in \mathbb{N}_0$.

Given the coefficient representation

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

of a polynomial $p(x)$, we want to evaluate $p(x)$ at each of a set $E_m := \{\alpha_i \mid 0 \leq i < m\}$ of m distinct evaluation points.

Idea:

Reduce the problem of evaluation of one polynomial $p(x)$ of degree n to the evaluation of two polynomials $p_1(y)$ and $p_2(y)$ of degree $\frac{n-1}{2}$ followed by one multiplication and one addition.

$\rightsquigarrow$

$$\begin{aligned} p(x) &= a_0 + a_1 x^1 + a_2 x^2 + \dots + a_n x^n \\ &= (a_0 + a_2 x^2 + a_4 x^4 + \dots + a_{n-1} x^{n-1}) \\ &\quad + (a_1 x^1 + a_3 x^3 + \dots + a_n x^n) \\ &= (a_0 + a_2 x^2 + \dots + a_{n-1} x^{n-1}) + x(a_1 + a_3 x^2 + \dots + a_n x^{n-1}) \end{aligned}$$

After substitution of y for x^2 we obtain

$$= (a_0 + a_2 y + \ldots + a_{n-1} y^{\frac{n-1}{2}}) + x(a_1 + a_3 y + \ldots + a_n y^{\frac{n-1}{2}})$$

$$= p_1(y) + x \cdot p_2(y)$$

where $p_1(y)$ and $p_2(y)$ are polynomials of degree $\frac{n-1}{2}$.

$\Rightarrow$

We can evaluate $p(x)$ at the points $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ by evaluation of $p_1(y)$ and $p_2(y)$ at the points $\beta_0, \beta_1, \ldots, \beta_{m-1}$ with $\beta_i = \alpha_i^2$, $0 \leq i < m$ followed by one multiplication and one addition for each evaluation point β_i, $0 \leq i < m$.

Question:
What would be the total number of arithmetic operations if we apply the reduction above recursively?

Answer:
$\Omega(m \cdot n)$ since we need one multiplication and one addition for each subproblem for each evaluation point.

Assume that we can use the evaluation points in such a way that during each recursion step the number of evaluation points would be halved.

$\Rightarrow$

Let $T(2^s)$ denote the total number of arithmetic operations used for the evaluation of $p(x) = \sum_{i=0}^{n} a_i x^i$ at $m = 2^s$ evaluation points.

⇝

We obtain the following recursion equality:

$$T(2^s) = 2\,T(2^{s-1}) + 2 \cdot 2^s$$

where the last term $2 \cdot 2^s$ counts for each evaluation point one multiplication and one addition.

Since $T(1) = 0$ the recursion equality has the following solution

$$T(2^s) = 2 \cdot 2^s \cdot s.$$

Hence,

$$T(m) = O(m \cdot \log m).$$

⇝

Goal:

Construction of a set of evaluation points with the property that squaring all evaluation points halves the number of distinct evaluation points.

We cannot construct such a set of evaluation points within the real numbers. But we can do this within the complex numbers.

A complex number $\omega \in \mathbb{C}$ is a <u>primitive kth root of unity</u> if

i) $\omega^k = 1$ and

ii) $\omega^j \neq 1$ for all $0 < j < k$.

<u>Example 2.3</u>

For $k \in \mathbb{N}$ we define

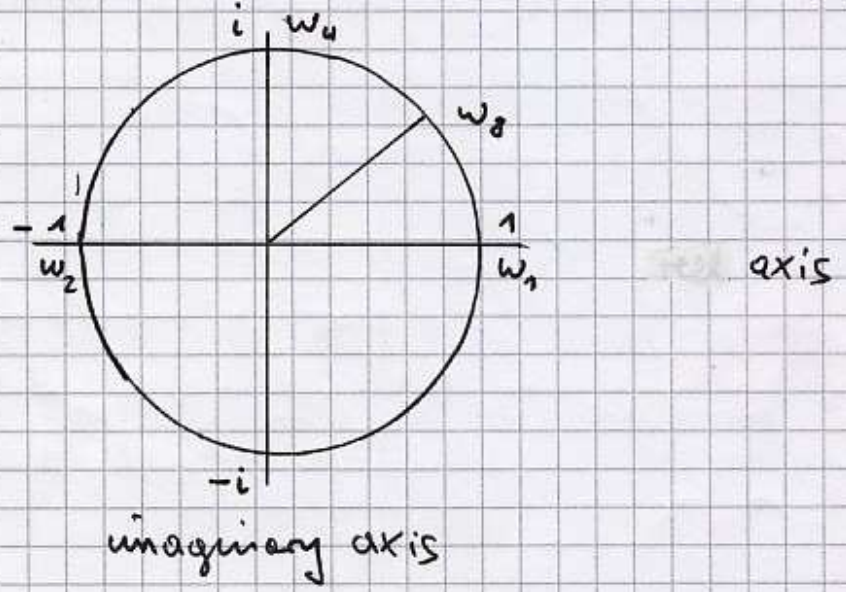
$$\omega_k := e^{\frac{2\pi i}{k}}.$$

ω_k is a primitive kth root of unity; i.e.,

$\omega_k^k = 1$ and $\omega_k^j \neq 1$ for $0 < j < k$.

For example we obtain

$\omega_1 = 1$, $\omega_2 = -1$, $\omega_4 = i$ and $\omega_8 = \frac{1+i}{\sqrt{2}}$



Note that always $\omega_{2k}^2 = \omega_k$.

Lemma 2.4

For $\omega = \omega_k$ these holds

$$\sum_{0 \le s < k} \omega^{s\alpha} = \begin{cases} k & \text{if } \alpha \equiv 0 \bmod k \\ 0 & \text{otherwise.} \end{cases}$$

Proof:

If $\alpha \equiv 0 \bmod k$ then by definition

$$\omega^{s\alpha} = 1 \quad \text{for } 0 \le s < k$$

and hence,

$$\sum_{0 \le s < k} \omega^{s\alpha} = \sum_{0 \le s < k} 1 = k.$$

Assume that $\alpha \not\equiv 0 \bmod k$.

Note that

$$z := \omega^\alpha \neq 1 \text{ and } z^k = \omega^{\alpha k} = 1^\alpha = 1.$$

$\Rightarrow$

$$\sum_{0 \le s < k} \omega^{s\alpha} = \sum_{0 \le s < k} z^s = \frac{z^k - 1}{z - 1} = 0$$

■

Let ω be a kth root of unity. Then we choose the following evaluation points:

$\xi_i := \omega^i, \quad 0 \le i < k.$

Example 2.4

Let $k = 2^3$. Then $\omega = \omega_8$.

$\Rightarrow$

$\xi_j = \omega^j$ for $0 \le j < k$.

Then $\omega^2 = \omega_4$ implies

$$\varphi_j = \xi_j^2 = \omega^{2j} = \omega_4^j.$$

◆

If ω is a $2k$th (primitive) root of unity then ω^2 is a kth root of unity. Hence, after squaring all evaluation points ω^i, $0 \le i < 2k$ the number of distinct evaluation points is halved.

$\Rightarrow$

The chosen set of evaluation points has the desired property.

2.2.3 The inverse transform: Fast interpolation

Again, let $n+1 = 2^r$ for some $r \in \mathbb{N}_0$. Let

$$p(x) = \sum_{i=0}^{n} a_i x^i.$$

Let ω be an $(n+1)$th primitive root of unity.

Assume that the values

$$y_i := p(\omega^i), \quad 0 \leq i \leq n$$

are given.

Goal:

To prove that the interpolation with respect to the Fourier points $\{ \langle \omega^j, y_j \rangle \mid 0 \leq j \leq n \}$ is equivalent to the performance of an appropriate forward transformation.

For reaching this goal let us consider the evaluation of the polynomial $p(x)$ at the set of points $\{ \omega^j \mid 0 \leq j \leq n \}$ as a vector-matrix product. Then we obtain:

$$(a_0, a_1, \dots, a_n) \cdot \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^n \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^n & \omega^{2n} & \dots & \omega^{n^2} \end{pmatrix} = (y_0, y_1, \dots, y_n)$$

Let V be the Vandermonde matrix above. Then we can write

$$(a_0, a_1, \dots, a_n) = (y_0, y_1, \dots, y_n) \cdot V^{-1}$$

Goal: Development of V^{-1}

Let us consider

$$\tilde{V} = (\tilde{v}_{ij})_{n+1,n+1},$$

where

$$\tilde{v}_{ij} := \frac{1}{n+1} \cdot w^{-ij}.$$

We number the rows and columns of the matrix $\tilde{V}$ from 0 to n.

We shall prove that $V^{-1} = \tilde{V}$. For obtaining this let us consider the product matrix $V \cdot \tilde{V}$. Then we obtain

$$(V\tilde{V})_{ij} = \frac{1}{n+1} \sum_{0 \le s \le n} w^{is} \cdot w^{-sj}$$

$$= \frac{1}{n+1} \sum_{0 \le s \le n} w^{s(i-j)}$$

$$\underset{\text{Le. 2.4}}{=} \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j. \end{cases}$$

$\Rightarrow$

$V\tilde{V}$ is the unit matrix and hence, $\tilde{V} = V^{-1}$.

Hence we obtain

$$(a_0, a_1, \dots, a_n) = (y_0, y_1, \dots, y_n) \cdot V^{-1}$$

$$= (y_0, y_1, \dots, y_n) \cdot \frac{1}{n+1} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w^{-1} & w^{-2} & \dots & w^{-n} \\ 1 & w^{-2} & w^{-4} & \dots & w^{-2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & w^{-n} & w^{-2n} & \dots & w^{-n^2} \end{pmatrix}$$

$$= \frac{1}{n+1} \cdot \left(\sum_{i=0}^{n} y_i z_0^i, \sum_{i=0}^{n} y_i z_1^i, \ldots, \sum_{i=0}^{n} y_i z_n^i \right),$$

where $z_j := \omega^{-j}$.

Note that

ω primitive $(n+1)$th root of unity

$\Rightarrow$

ω^{-1} is a primitive $(n+1)$th root of unity.

$\Rightarrow$

The interpolation with respect to

$$\langle \omega^0, y_0 \rangle, \langle \omega^1, y_1 \rangle, \ldots, \langle \omega^n, y_n \rangle$$

is equivalent to the forward transformation with respect to the polynomial

$$\sum_{i=0}^{n} y_i x^i$$

at the points $(\omega^{-1})^0, (\omega^{-1})^1, \ldots, (\omega^{-1})^n$.