DCB-Script

by Anna Feldmann

Contents

1 Graph Theory - an Introduction

1.1 Basic Definitions

Definition 1.2. A (simple) graph $G = (V, E)$ is a tupel with a non-empty set V (vertices) and unordered pairs of distinct vertices $(u, v) \in E$

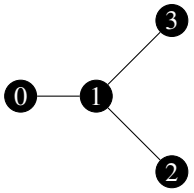


Figure 1.1: A simple graph $G = (V, E)$ with $V = \{0, 1, 2, 3\}$ and $E = \{(0, 1), (1, 2), (1, 3)\}$.

Definition 1.3. Two vertices $u, v \in V$ are adjacent iff $(u, v) \in E$.

Definition 1.4. Vertex v is incident to an edge e iff $\exists\, e = (x, v) \in E$.

Definition 1.5. A directed graph $G = (V, E)$ has a non-empty vertex set V and ordered tupels $[u, v]$.

Figure 1.2: A directed graph $G = (V,E)$ with $V = \{0,1,2\}$ and $E = \{[0,1],[1,2],[2,1]\}$.

Definition 1.6. The vertex degree of a vertex is defined as follows:

- undirected case
  - $deg(v) = |\{e \in E \mid v \in e\}|$
- directed case
  - $deg_{out}(v) = |\{e \in E \mid e = [v,x]\}|$
  - $deg_{in}(v) = |\{e \in E \mid e = [x,v]\}|$
  - $deg(v) = deg_{in}(v) + deg_{out}(v)$

1.7 Theorems

Theorem 1.1. *For any (undirected) graph $G = (V,E)$ holds*

$$\sum_{v \in V} deg(v) = 2\,|E|$$

Proof. Summing the degree of each vertex can be regarded as counting the number of incident edges for each vertex. Since an edge always connects two vertices, each edge is counted twice. □

Theorem 1.2. *Every Graph $G = (V,E)$ contains an even number of vertices with odd degree.*

Proof. See exercises. □

Theorem 1.3. *Let $G = (V,E)$ be an undirected graph, then G has two vertices of the same degree.*

Proof. See exercises. □

1.8 Further Definitions

Definition 1.9. Let G_1, G_2 be two graphs. Then G_1 is isomorphic to G_2 ($G_1 \cong G_2$) if there is a bijective mapping $\phi : V(G_1) \to V(G_2)$ s.t. $\forall (u,v) \in E(G_1) \Leftrightarrow (\phi(u), \phi(v)) \in E(G_2)$.

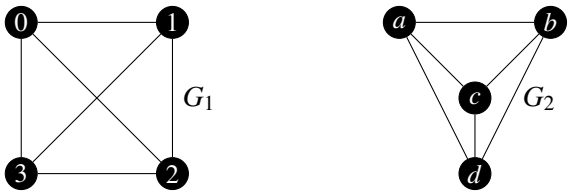


Figure 1.3: The graph G_1 is isomorphic to G_2, since there exists a bijective mapping $\phi : V(G_1) \mapsto V(G_2)$ s.t. $\phi(0) = a, \phi(1) = b, \phi(2) = c, \phi(3) = d$ and $\forall (u,v) \in E(G_1) \Leftrightarrow (\phi(u), \phi(v)) \in E(G_2)$.

Definition 1.10. A subgraph of a graph can be defined in two ways.

1. A graph H is subgraph of a graph G $H \subseteq G$ if

   - $V(H) \subseteq V(G)$
   - $E(H) \subseteq E(G)$

2. An arbitrary graph F is a subgraph of G if

$$\exists H \subseteq G : F \cong H$$

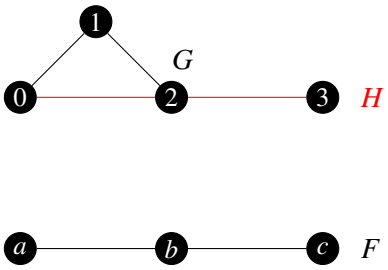


Figure 1.4: According to definition 1 F is not a subgraph of G but it is according to definition 2.

Definition 1.11. If all pairs of vertices of F with $F \cong H \subseteq G$ that are adjacent in G via a mapping ϕ are also adjacent in H (and thus in F), we call F with respect to H an induced subgraph.

Definition 1.12. A subgraph $H \subseteq G$ is a spanning subgraph if $V(H) = V(G)$.

Definition 1.13. A complete graph G is denoted by K_n i.e. all vertices of $V(K_n)$ are pairwise adjacent.

Definition 1.14. A walk in a graph $G = (V,E)$ is a sequence of vertices $(v_1, v_2, \ldots, v_n)$ s.t $(v_i, v_{i+1}) \in E$.

Figure 1.5: F is not an induced graph of G as $e = (0,2)$ is missing.

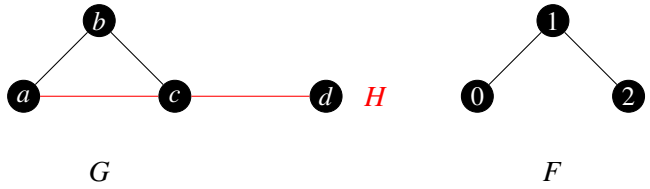


Figure 1.6: F is an induced graph of G as $F \cong H \subseteq G$.

Definition 1.15. A closed walk is a walk where in addition there exists an edge between start and end point, thus a closed walk is a sequence of vertices $(v_1, v_2, \ldots, v_n, v_1)$.

Definition 1.16. A path is a walk where all vertices and hence all edges are distinct.

Definition 1.17. A cycle is a path s.t. $(v_1, \ldots, v_n, v_1)$, i.e. only the first and last vertices are identical.

Definition 1.18. A graph $G = (V,E)$ is connected if for any two vertices $u, v \in V$ there exists a path connnecting u and v. Otherwise G is disconnected.

Definition 1.19. Let $G = (V,E)$ be a graph. $H \subseteq G$ is a connected component of G if

$$\nexists H' \subseteq G\colon\ H' \text{ is connected} \wedge H \subsetneq H'$$

Definition 1.20. A tree is a connected graph that does not contain cycles.

Definition 1.21. A forest $F = (V,E)$ is a graph where each connected component is a tree.

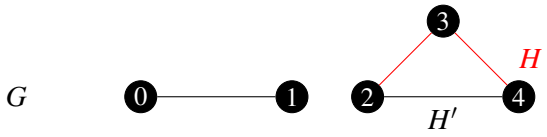


Figure 1.7: H is not a connected component of G since $H' = (\{2,3,4\}, \{(2,3),(3,4),(2,4)\}) \subseteq G$ and obviously, $H \subsetneq H'$.

1.22 More Theorems

Theorem 1.4. *T is a tree if and only if there exists only one path between any two vertices $u, v \in V(T)$*

Proof.

Let T be a tree. Since T is connected there exists at least a path between u and v with $u, v \in V$. Assume there are two different paths, so they differ at least in one vertex w. Then there exists a cycle $(w, \ldots, u, \ldots \tilde{w}, \ldots, u, \ldots, w)$, this contradicts the definition of a tree.↯
Let there exists exactly one path between u and v, thus T is connected. Assume there exists a cycle. Then there exist two paths between u and v, which is a contradiction to our premise.↯

□

Theorem 1.5. *Let $G = (V, E)$ be a connected graph. G is a tree if and only if $|E| = |V| - 1$.*

Proof. Let G be a tree. We perform a proof by induction over the number of vertices $|V|$= n.

- **Induction Basis**
  Let $n = 1$.
  $|E| = 0 = |V| - 1 = 1 - 1 = 0$.
- **Induction Hypothesis**
  Let k be an integer, assume the hypothesis is true for trees with $|V| \leq k$.
- **Inductive Step**
  Consider T with $|V| = k$. By removing an arbitrary edge $(u, v) \in E$ we obtain a tree $T' = T \setminus \{(u, v)\}$ which is disconnected. Thus T' has exactly two connected compounds T_1, T_2 that are trees with the following properties:
  - $|V(T_1)| = k_1 < k$
  - $|V(T_2)| = k_2 < k$
  - $k_1 + k_2 = k$
  - $|E(T_1)| = k_1 - 1$
  - $|E(T_2)| = k_2 - 1$

Now the number of edges in T can be computed as follows:

$$\begin{aligned} |E(T)| &= |E(T_1)| + |E(T_2)| + 1 \\ &= k_1 - 1 + k_2 - 1 + 1 \\ &= k_1 + k_2 - 1 \\ &= k - 1 \end{aligned}$$

To show the other direction, i.e. $|E| = |V| - 1 \Rightarrow G$ is a tree, we assume that G fulfills $|E| = |V| - 1$ and is connected. If G is not a tree, it has to contain cycles. Now, we can remove all edges from these cycles such that the modified graph $G' = (V, E')$ becomes a tree, however G' is still connected. Let us assume that k such edges have been removed from G. Then we have $|E'| = |V| - 1 - k$ which is a contradiction to the fact that G' is still connected. ↯

□

Corollary 1.6. *If G is connected, then $|E| \geq |V| - 1$ and G has a spanning tree.*

1.23 Some more definitions

Definition 1.24. Let $G = (V, E)$ be a graph. The distance between two arbitrary vertices $u, v \in V$ is the number of edges involved in a shortest path connecting u and v. If such a path does not exist, we set the distance between u and v to infinity, i.e. $d(u, v) = +\infty$.

Definition 1.25. The graph $G = (V, E)$ is a weighted graph if there exists a weighting function $w : E \to \mathbb{R}_0^+$ that assigns a weight to each edge.

Definition 1.26. The length of a path $P = (u, \ldots, v)$ in a weighted graph $G = (V, E)$ is given by $\sum_{e \in P} w(e)$. To compute the length of a path in a non-weighted graph $G = (V, E)$, we set $w(e) = 1$ for all edges in G. The length of a non-existing path is set to infinity.

Lemma 1.7. *Connected subgraphs of shortest paths (subpaths) are itself shortest paths.*

Proof. See exercise. □

Lemma 1.8. *Let $G = (V, E)$ be a graph and $u, v, x \in V$. Then the triangle inequality holds:*

$$d(u,v) \leq d(u,x) + d(x,v)$$

Proof. Let us assume that we have $d[u,v] > d[u,x] + d[x,v]$. But this would imply that the shortest path between u and v is actually not the shortest path since the sub-paths $[u,x]$ and $[x,v]$ are actually shorter. Due to this contradiction the triangle inequality holds. □

1.27 Graph Representations

There are several possibilities to represent a graph:

1. Adjacency matrix: a $|V| \times |V|$ matrix with

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{else} \end{cases}$$

2. Incidence matrix: a $|V| \times |E|$ matrix with

$$B_{ij} = \begin{cases} 1 & \text{if } i \text{ is incident to edge } j \\ 0 & \text{else} \end{cases}$$

3. Adjacency list: a list L in which each element is a set of nodes

$$L[i] = \{v | (v,i) \in E\}$$

Advantage of the Adjacency Matrix

The adjacency matrix can be used to compute the number of walks of length k between any two vertices $v_i, v_j \in V$ in a graph $G = (V,E)$. The number of k-walks for a node is given by

$$A_{ij}^k = (\underbrace{A \cdot A \cdot \ldots \cdot A}_{k \text{ times}})_{ij}$$

1.28 Computing the Distance between Vertices

We know that $d(u,v)$ is the distance between the nodes $u,v \in V$, which is defined as the number of edges on a shortest path between u and v. In an undirected graph, the distances of a node v to all other nodes x can be determined using breadth first search (BFS), see algorithm **??**

Algorithm 1 Bread - First - Search

Require: graph $G = (V,E)$, start node $s \in V$

Ensure: arrays of distances δ and predecessors *pred*

```
for all v ∈ V do
    δ[v] ← ∞
    pred[v] ← NIL
end for
δ[s] = 0
enqueue(Q, s)
while Q is not empty do
    u ← dequeue(Q)
    for all v ∈ adj(u) do
        if δ[v] = ∞ then
            δ[v] ← δ[u] + 1
            pred[v] = u
            enqueue(Q, u)
        end if
    end for
end while
```

Theorem 1.9. *$BFS(G,s)$ needs $O(|V|+|E|)$.*

Proof. See exercise. Hint: use $\sum_{v\ inV} deg(v) = 2|E|$. □

Theorem 1.10. *When $BFS(G,s)$ terminates $\delta(v) = d(s,v)\ \forall v \in V$.*

1.29 Graph Coloring

1.29.1 Definition

Definition 1.30. A (proper) coloring γ of a graph $G = (V,E)$ is a mapping γ from the set of vertices V to a set of colors C, i.e. $\gamma : V \to C$. A coloring is called proper when the coloring function γ fulfills

$$\forall\ (v_i, v_j) \in E : \gamma(v_i) \neq \gamma(v_j)$$

This means that all pairwise adjacent vertices need to exhibit alternating colors.
For a trivial coloring one could just use $|C| = |V|$, that is, assign a different color to each vertex. But, usually one wants to determine the minimal number of colors one needs for a coloring of G. A coloring γ that requires at most k colors is called a k-coloring of the graph G, that is, $|C| \leq k$. The coloring that uses the smallest k possible is called the chromatic number $\chi(G)$ of the graph G.

1.30.1 Scheduling Problem

Consider a set of courses, each requiring one time unit. The problem now is to find a scheduling of courses requiring the fewest time units in total such that courses c_i and c_j do not occur at the same time if a student wants to visit both. In the graph each vertex v_i represents the course c_i, conflicting courses are connected with an edge. Now, the chromatic number of the graph gives the minimal number of timeslots for the scheduling problem.
Graph coloring can also be used to model the Sudoku game, using vertices to represent cells and edges to represent dependencies.
Hint: keep graph coloring in mind for exercises regarding NP-complete problems.

Theorem 1.11. *For any graph $G = (V,E)$ we have*

$$\chi(G) \leq 1 + \Delta(G)$$

where $\Delta(G)$ is the maximal degree of any node in G. The equality

$$\chi(G) = 1 + \Delta(G)$$

holds for complete graphs.

Proof. This proof is based on induction over $|V|$.

- **Induction Basis**
  Let $|V| = 1$. Then we have $\chi(G) = 1$ and $\Delta(G) = 0$ and therefore the equation $\chi(G) \leq 1 + \Delta(G) = 1 = 1 + 0 = 1$ holds.
- **Induction Hypothesis** The assumption $\chi(G) \leq 1 + \Delta(G)$ holds for all graphs with $|V| \leq k$ where k is an integer.
- **Induction Step** Let $G = (V,E)$ be a graph with $|V| = k$ and $v \in V$ an arbitrary vertex. Now, consider the graph $G - v$, which is defined as $G = (V \setminus \{v\}, E \setminus \{(x,v) | \forall x \in V\})$. Then we obviously have $|V(G-v)| < k$. As a consequence we can use the induction hypothesis and have $\chi(G-v) \leq 1 + \Delta(G-v)$, which means that there exists a $\Delta(G-v)+1$-coloring. We also know that the node v had at most $\Delta(G)$ neighbours, that is

  $$\Delta(G-v) \leq \Delta(G)$$

  We can now distinguish two cases:

  1. $\Delta(G-v) = \Delta(G)$
     This implies that $\chi(G) \leq 1 + \Delta(G-v) = 1 + \Delta(G)$ and the assumption holds.
  2. $\Delta(G-v) < \Delta(G)$
     In this case v was the vertex with maximal degree. Thus the new maximal degree $\Delta(G-v)$ is at most $\Delta(G) - 1$ and we have following implication:

     $$\Delta(G-v) < \Delta(G) \Rightarrow 1 + \Delta(G-v) \leq \Delta(G)$$

     This implies that $\chi(G) \leq 1 + \Delta(G)$ holds.

□

1.30.2 Bipartitions

Definition 1.31. A bipartion of the vertex set V is the the disjoint union of the sets V_1 and V_2 where $V = V_1 \dot{\cup} V_2 = V_1 \cup V_2 \wedge V_1 \cap V_2 = \emptyset$. A graph $G = (V,E)$ is called bipartite if and only if there exists a

partition of V in V_1 and V_2 s.t.

$$V = V_1 \dot{\cup} V_2$$
$$\forall e = (a,b) \in E : a \in V_1, b \in V_2$$

A graph is bipartite if and only if a 2-coloring exists.

Theorem 1.12. *A graph $G = (V,E)$ is bipartite if and only if G does not contain any odd cycles.*

Proof. Let G be bipartite. Assume for contradiction that the graph $G = (V,E)$ contains at least one odd cycle. Let $c = (v_1, \ldots, v_n, v_1)$ be one such cycle of length n, where n is odd, that is, $n = 2i+1$, where $i \in \mathbb{N}$. The vertices in the cycle c have to be partitioned in the following way:

- $v_l \in V_1$ if l is odd
- $v_l \in V_2$ if l is even

Since n is odd, we know that $v_1, v_n \in V_1$. Hence, G cannot be bipartite as there exists the edge (v_n, v_1). $\lightning$

Suppose that G does not contain any odd cycles. Let us further assume, without loss of generality, that G is connected and $|V| \geq 2$. Now, let us fix $v \in V$ and define

$$X = \{x \in V | d(v,x) = 2n, n \in \mathbb{N}\}$$
$$Y = \{y \in V | d(y,x) = 2n+1, n \in \mathbb{N}\}$$

where X is the set of even-distance vertices to v and Y is the set of odd-distance vertices. We now claim that X and Y are the bipartition sets, i.e. $X \dot{\cup} Y = V$ and that for all edges (u,w) we have $u \in X$ and $w \in Y$ or vice versa. Assume that there exist $x_1, x_2 \in X$ with $(x_1, x_2) \in E$. Let us now consider we have a node v' such that $d(v,v') = k$ and there is an edge (v', x_1) and (v', x_2). By construction we have the following properties:

- $d(v,x_1) = 2n_1$
- $d(v,x_2) = 2n_2$
- $d(v',x_1) = 2n_1 - k$
- $d(v',x_2) = 2n_2 - k$

Now, we have to consider two cases.

1. $d(v,v') = k$ is even.
   Then $d(v',x_1)$ and $d(v',x_2)$ are even and we have an odd cycle $c = (v',\ldots,x_1,x_2,\ldots,v')$ ↯

2. $d(v,v') = k$ is odd.
   Then $d(v',x_1)$ and $d(v',x_2)$ are odd and consequently $d(v',x_1)+d(v',x_2)$ is even and we have again an odd cycle $c = (v',\ldots,x_1,x_2,\ldots,v')$ ↯

□

2

Complexity

2.1 Introduction to Complexity

A very important question is whether all problems are solvable or not. In 1936 Adam Turing proved that not all problems are solvable with the Halting Problem.

2.1.1 The Halting Problem

The problem is as follows.
Is there an algorithm that decides whether any other program terminates or runs forever?
The proof is done via contradiction and we will give here only the proof idea. Assume that such an algorithm `HALT` exists, which takes another algorithm `A` as an input and is able to determine whether this algorithm terminates or not. Now let us consider algorithm `A` as:

Require: /
Ensure: NIL

while HALT(test()) **do**
  pass
end while

We can now distinguish two cases:

1. `test()` terminates
2. `test()` does not terminate

If `test()` terminates, `HALT` has to return true. But due to the while-loop in `test()`, `test()` does not terminate. which is a contradiction. If we assume that `test()` does not terminate, then `HALT` would return *FALSE*, which would lead to the termination of `test()`, which is a contradiction. Therefore, the halting problem is undecidable.

2.2 Decision vs. Optimization Problems

Generally, one can differentiate between two types of problems: decision and optimization problems. Decision problems can always be answered in a yes/no fashion, while optimization problems output more intricate results.

Example of an Optimization Problem

Given a graph $G = (V, E)$ and two nodes $u, v \in V$, determine the shortest path between u and v.

Example of a Decision Problem

Given a graph $G = (V, E)$, two nodes $u, v \in V$ and an integer k, determine if there is a shortest path between u and v that is at most of length k.

Optimization and decision problems are related due to the fact that a simple optimization problem will also be easily (polynomial) solvable in its decision form.

2.3 The sets P and NP

Definition 2.4. P:= set of all decision problems that can be solved in worst-case polynomial time $O(n^a)$.

Definition 2.5. NP:= set of all decision problems that can be "verified" in polynomial time. Thus given a solution, one can verify if the solution is correct or not. NP stands for non-deterministic polynomial. Obviously $P \subseteq NP$.

Definition 2.6 (Deterministic Machine)**.** For every input, every step is uniquely determined

Millenium problem Is $NP = P$ or $NP \neq P$?

Polynomial reduction Given a problem A with unknown complexity. If we can map an arbitrary instance $\alpha \in A$ in polynomial time onto an arbitrary instance $\beta \in B$ with B being a problem with polynomial complexity and the decision (yes/ no) given by β is also a solution for α, then we have shown that A is easy by the easiness of B.

$$B \in P \Rightarrow A \in P := A \leq_p B$$

2.7 NP-completeness

2.7.1 How to prove NP-completeness

Definition 2.8. A problem D is NP-complete if and only if

1. $D \in NP$
2. D is NP-hard:
   $\forall D' \in NP$: $D' \leq_p D$
   D is at least as hard as any other problem of $D' \in NP$

Steps for proving NP-completeness of a problem D

1. Show D *inNP*
2. Show D is NP-hard
   (a) Pick one problem $D' \in NP \wedge D'$ NP-complete.
   (b) Find a mapping/transformation that maps any instance $d' \in D'$ to an instance $d \in D$.
   (c) Show that the transformation can be done in polynomial time.
   (d) Show that $d' \in D' = yes \Leftrightarrow d \in D = yes$.

2.8.1 Transitivity of $\leq_p$

If D' is NP-complete (NPC) then all $D'' \in NP$ can be mapped to D' in polynomial time.

$$forall\, D'' \in NP\colon D'' \leq_p D' \wedge D' \leq_p D$$

Thus $\forall\, D' \in NP$ there exists a mapping from $D' \rightarrow D$ in polynomial time.

2.8.2 SAT-Problem (satisfiability problem)

The satisfiability problem is the first known example of an NP-complete problem.

Definition 2.9. Given a boolean formula with literals and `AND, OR, NOT` operators.
Question: Is there some assignment of `TRUE-` or `FALSE-` values to the variables $x_1, \ldots, x_n$ s.t. the formula becomes true.

Proof of NP-completeness for SAT

1. $SAT \in NP$
2. Then it was shown that any problem $D \in NP$ is in polynomial time reducible to SAT, see *Cook-Levin theorem*.

Conjunction normed form (CNF)

Every boolean formula can be rewritten in a so called conjunctive normed form (CNF), where conjunctive means a conjuction of clauses. In this context a clause is defined as disjunction of literals where a literal and its negation can not be in the same clause. Example:

$$\underbrace{\overbrace{(x_1 \vee x_2)}^{\text{clause}} \wedge \overbrace{(x_1 \vee \neg x_3)}^{\text{clause}}}_{\text{conjunction}}$$

k-SAT

Every clause contains exactly k literals. We know that $2-SAT \in P$.

Theorem 2.1. *$3-SAT$ is NP-complete.*

Proof.

1. $3-SAT \in NP$
2. $3-SAT$ is NP-hard as $SAT \leq_p 3-SAT$
   Given an arbitrary instance $C_1,\ldots,C_n$ of SAT of SAT in CNF. There are four $k-SAT$ possibilities, we have to consider:
   a) Let $|C_i| = 3$, then no transformation is needed.
   b) Let $|C_i| = 2$. Let $C_i = x \vee y$. We introduce a new variable u, s.t we can transform C_i into $C_i^{'} = x \vee y \vee u$ and $C_i^{''} = x \vee y \vee \neg u$. If C_i is true then either x or y are true and thus $C_i^{'}$ and $C_i^{''}$ are true. On the other hand if $C_i^{'}$ and $C_i^{''}$ are true, then the choice of u is arbitrary and at least one of the variables x and y has to be true as well.
   c) Let $|C_i| = 1$. Then we denote $C_i = x$ and we introduce an arbitrary literal u to transform C_i into $C_i^{'} = x \vee u$ and $C_i^{''} = x \vee \neg u$. With the same reasons as above we can proof this case, and for $|C_i| = 2$ we have already shown the transformation.
   d) Let $|C_i| > 3$, we can denote C_i as $(x_1 \vee x_2 \vee \ldots \vee x_k)$. We introduce u_{k-3} new literals and transform C_i to:

$$(x_1 \vee x_2 \vee u_1) \wedge (x_3 \vee \neg u_1 \vee u_2) \wedge \ldots \wedge (x_{k-2} \vee \neg u_{k-4} \vee u_{-3}) \wedge (x_{k-1} \vee x_k \vee \neg u_{k-3})$$

   If C_i is true then at least one of the x_i must be true. Let u_j be true up to the point where x_i was accounted and u_j false thereafter. Assume we have given such a *cascade* of clauses of $3-SAT$. We have to show that if the cascade is true then $C_i \in SAT$ is true. In the following we will proof the contraposition and thus the assumption. Assume we have $C_i = (x_1,\ldots,x_k)$ which is not satisfiable then all x_i must be false. We will perform the proof by contradiction. Assume that the cascade of clauses is satisfied (iff each of the clauses is true) and all x_i are false. We look at the clauses beginning in the end.$(x_{k-1} \vee x_k \vee \neg u_{k-3})$ has to result in true and as all x_i are false $\neg u_{k-3}$ has to be true, and thus u_{k-3} is false. But then in $(x_{k-1} \vee \neg u_{k-4} \vee u_{k-3})$ $\neg u_{k-4}$ has to be true $\Leftrightarrow$ u_{k-4} is false, which is only allowed to be if x_{k-1} would have been true, which it is not. $\lightning$
   If all u_i are false then obviously $(x_1 \vee x_2 \vee u_1) = (0 \vee 0 \vee 0) = 0$. But then then C_i is not satisfiable $\lightning$.

□

2.9.1 Clique-Problem

Definition 2.10 (Clique)**.** Let $G = (V,E)$. A clique $C = (V',E')$ is a complete subgraph of G of the form $\forall\, u,v \in V' \colon (u,v) \in E'$. The size of a clique is denoted as $|V'|$.

Definition 2.11 (Clique-Problem)**.** Given a graph G find a clique of size k. We have $\binom{V}{k}$ possible combinations with $k \leq \frac{V}{2}$.

Theorem 2.2. *The Clique-Problem is NP-complete.*

Proof.

1. Clique $\in NP$ as for the verification we only need to consider $\binom{V'}{2}$ which lies in $O(V'^2)$.
2. We show that $3-SAT \leq_p Clique$.
   Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ and $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ representing the vertices v_1^r, v_2^r, v_3. We connect to vertices v_i^r, v_j^s if both holds:

   a) $r \neq s$

   b) $l_i^r \neq \neg l_j^s$

   Assume ϕ is satisfiable then each clause C_r contains at least one true literal l_i^r that is true. Pick for each C_r one true literal l_i^r (pick corresponding vertices in G). We have chosen k vertices = V'. Claim that $\langle V' \rangle$ is a clique. Let $v_i^r, v_j^s \in V' \wedge r \neq s$, both corresponding literals are mapped to be true. l_i^r, l_j^s cannot be the negation of each other. Hence, v_i^r and v_j^s are connected. Assume G has a clique $\langle V' \rangle$ of size k. Then no edges connect vertices of the same class. V' contains exactly a vertex per clause. We assign true to each literal l_i^r if the corresponding vertex is in the clique. Hence, each clause gets true and thus ϕ is true.

□

2.11.1 Various NP-complete Problems

Decision Problem: Clique-Problem

Given a graph G and an integer k.
Question: Is there a clique in G of size k.

Decision Problem: Vertex - Cover - Problem (VCP)

Definition 2.12. A vertex cover of a graph $G = (V,E)$ is a subset $V' \subseteq V$ s.t. $\forall (u,v) \in E : u \in V' \vee v \in V'$

Optimisation problem Find a vertex cover of G of minimal size.

Decision Problem Given a graph G, integer k. Is there a vertex cover of size k?

Theorem 2.3. *The minimal vertex cover problem is NP-complete.*

Proof.

1. $VCP \in NP$
   Given a VC V' , where $|V'| = k$. Check for all edges (x,y), $x \in V' \vee y \in V'$
2. VCP is NP-hard. We show Clique $\leq_p VCP$.
   G has a clique $< V' >$ of size $l \Leftrightarrow \bar{G}$ has a VC of size $|V| - |V'| = k$. Assume G has a clique, induced by $V' \subseteq V$, $|V'| = l$. Claim $V - V'$ is VC of $\bar{G}$. Let $(u,v) \in \bar{E}$ ($(u,v) \notin E$). Suppose that (u,v) is not covered by VC, then $u,v \notin V - V' \Rightarrow u,v \in V' \Rightarrow (u,v) \in E$ ↯
   Let $V - V'$ be a VC of $\bar{G} \Rightarrow \forall (u,v) \in \bar{E} : u \in V - V' \vee v \in V - V' \Rightarrow$ at least one of u or v is not contained in V'. Hence, $x,y \in V' \Rightarrow (x,y) \notin \bar{E} \Rightarrow \forall x,y \in V' : (x,y) \in E \Rightarrow < V' >$ is a clique.

□

ILP - Integer Linear Programming

Definition 2.13. An integer liner programming problem has the following form:

$$\max c^t x$$

subject to

$$Ax \leq b \tag{2.13.1}$$

$$x \geq 0\,, x \in \mathbb{N} \tag{2.13.2}$$

Given integer k is there an assignment to variables $x = (x_1, \ldots, X_k)$, where $x_i \in (N)$ s.t

$$c^t x \geq k \qquad Ax \leq b \tag{2.13.3}$$

$$x \geq 0\,, x \in \mathbb{N} \tag{2.13.4}$$

Theorem 2.4. *Theorem ILP is NP-complete.*

Proof.

1. $ILP \in NP$ as we can obviously check a given solution in polynomial time.
2. Give an instance $x \in 3SAT$ map each x_i to a new variable $z_i \in \{0,1\}$. Let have for each clause $x_1 \vee \neg x_2 \vee x_3$ a transformation to $z_1 + (1 - z_2) + z_3$. So we can show that $\{0,1\}$ ILP is NP-complete.

□

Decision Problem $k - COL$

Given a graph $G = (V,E)$ and an integer $k \geq 3$. Is $\chi(G) = k$?

3

Some Molecular Biology

3.1 History

A challenge in molecular biology was to understand inheritance.

- In 1865, Mendel introduced an abstract, mathematical model of inheritance in which the basic unit of inheritance was represented by a gene.
- Several years later, in 1869, Johannes Friedrich Miescher first discovered the DNA from the nuclei of white blood cells, calling it nuclein.
- Only in 1952, the Hershey - Case - Experiments with T_2-Phage revealed that the genetic information is carried in the DNA and not in proteins.
- In 1953 James Watson and Francis Crick proposed the now famous double helical structure for DNA. In the corresponding paper, they already stated that " It has not escaped our notice that the specific pairing we have postulated immediately suggests a possible copying mechanism for the genetic material" .

3.2 DNA - deoxyribonucleic acid

DNA has the following properties:

- carries the information of an organism
- basis of hereditary
- is a polymer, large molecule of repeating structural units, made of small units the so called nucleotides Adenin (A), Cytosin (C), Thymin (T) and Guanin (G).

- has a double helical structure
- is complementary, that is A binds to T, and C to G.
- DNA is a word of an alphabet $\mathbb{A} = \{A, C, G, T\}$.

3.3 RNA

RNA has the following properties:

- it is a polymer
- instead of the nucleotide Thymin (T) RNA has the nucleotide Uracil (U)
- RNA is a word over the alphabet $\mathbb{A} = \{A, C, G, U\}$.
- A Protein is a word over alphabet $\mathbb{A} = \{20 letters\}$.

Figure 3.1 represents the central dogma of moelcular biology, that is that information is carried form DNA to RNA and from RNA to proteins.

3.4 Cracking the Genetic Code

Even though, it was known that DNA carries the genetic information and also its structure, it still remained a problem how the creation of 20 amino acids out of four bases was possible. Obviously, two bases were not enough, since they could only form 16 (4^2) amino acids. However, a genetic unit of three bases lead to 64 possible amino acids, which was known to be larger than the existing 20

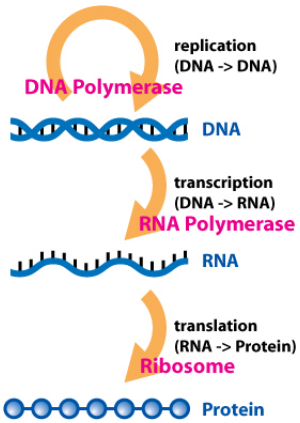


Figure 3.1: The central dogma of molecular biology.

amino acids. Thus, it remained still unclear how the genetic units were translated to proteins.

3.4.1 1954 - The diamond code (Gamow)

In the same year as Watson and Crick postulated the structure of the DNA, George Gamow, a russian physicist, introduced a purely formal (arbitrary) set of rules for the genetic code. The code was named after the diamond shape between four nucleotides in the DNA, which were encoding the proteins according to Gamow. Let 1,2 and 3 be three successive bases in the DNA on one strand, and 1' 2' and 3' their complementary bases on the other strand. Two opposite corners of the diamonds were now formed by 1 and 3' and 2 and 2', respectively. The order of the basis plays no role in the diamond code, that is a diamond rotated by 180Â° encodes for the same amino acid. Since the aim is to reveal how the 20 amino acids can be formed by the four bases, we will now count how many different diamonds, that is, amino acids, can be formed with the diamond code. The bases on position 1 and 3' are not related so we can choose arbitrary one of the four bases for each position. The opposite corners of the diamond represented by basis 2 and 2' are forming a base pair. Since there exist only two possible base pairings (A-T, C-G) and the order does not play a role, we have only two possibilities for these opposite corners. Now, demanding, that the bases on the 1 and 3' position are the same results in 8 possibilities. The problem to choose two distinct bases for the 1 and 3' position, is a combinatorical problem, that is, there are $\binom{4}{2}$ possibilities. In total, the diamond code achieved to show a code, which translate the four bases into 20 amino acids, introducing the "magic number twenty". All twenty possibilities are presented in Figure 3.2. The diamond code can be also seen as a triplet code, since it suffice to know one bases of the complement base pair in a diamond. Moreover,

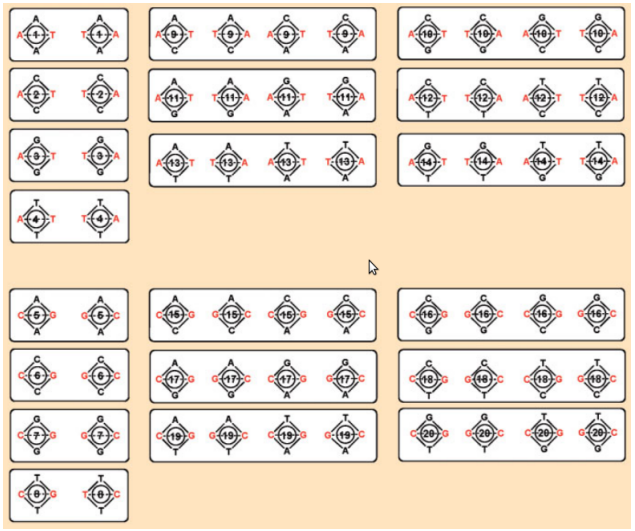


Figure 3.2: The 20 possible diamonds according to Gamow's theory.

the diamond code is an overlapping code, that is, each nucleotide appears in three adjacent codons (genetic units/ successive diamonds). The overlapping attribute was the downfall for the diamond code, since it is impossible to form all combinations of sequences where one aminoacid is enclosed by two identical amino acids, for example the amino acid sequences Ser-Tyr-Ser or Leu-Tyr-Leu cannot be formed by the diamond code. This limitation was found by Francis Crick. Sydney Brenner showed that no overlapping code is able to represent the proteins.

3.4.2 Crick's approach

In 1957, Francis Crick proposed a non-overlapping code,the so called comma free code. This implied that the reading frame of the triplets is determined by the triplets itself. A non-overlapping block of 3 letters was referred to as codon From the initial 64 possibilities, Crick excludes triplets consisting of only one bases, like AAA or UUU, since in the case of two successive triplets of such a form, the reading frame is not uniquely determined. The remaining 60 combinations can be grouped by three triplets, in which the bases are cyclic inverted, for example, ACA CAA and AAC. In each such a group, only one of the triplets is allowed to form an amino acid, otherwise the reading frame is not determined and we don't have a comma-free code. In total, we have then 20 groups, that is, we can form exactly the magic twenty amino acids. The 20 groups are shown in Figure 3.3.

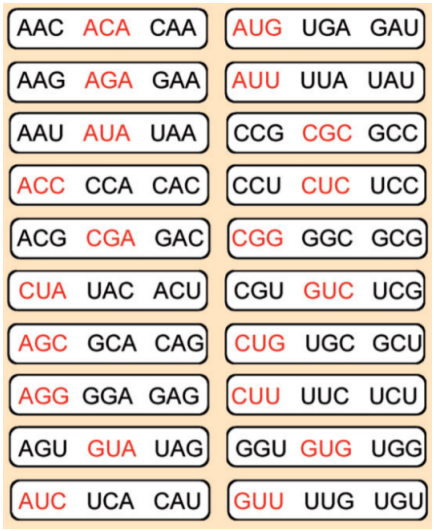


Figure 3.3: The 20 groups of amino acids according to Crick's comma-free code.

3.4.3 Nirenberg and Matthaei

Despite the elegance of the comma-free code, Marshall W. Nirenberg and J.Heinrich Matthaei disproved this approach with their experiments in 1961. In the experiments, they extracted the content of E.coli bacteria, and removed the original DNA with a DNAase. As a consequence, giving a synthetic mRNA to the extract, the corresponding protein will be created via the proteinbiosynthesis. They started their experiments with a poly-U chain, which resulted in a sequence of phenylalanin amino acids. This was the downfall of Crick's comma-free code, since these kind of codons were not allowed. Comparing the computing bases distribution in the codons and the relative amount of the formed amino acid enabled the mapping from amino acids to certain nucleotide combinations but not the right order of it. This was only achieved five years later. However, they could reveal some of the assignments with the approach described below:

1. UUUUUUUUU: $P_1 = \{\text{Phe}\}$, $B_1 = \{\text{ UUU}\} \Rightarrow$ UUU codes for Phe.
2. UGUGUGUGU: $P_2 = \{\text{Cys, Val}\}$, $B_2 = \{\text{ UGU, GUG}\} \Rightarrow$ UGU, GUG are coding for Cys and Val, but so far we don't know the right assignment.
3. UUGUUGUUG: $P_3 = \{\text{Leu, Val, Cys}\}$, $B_3 = \{\text{ UUG, UGU, GUU}\} \Rightarrow$ Since $P_2 \cap P_3 =$Cys and $B_2 \cap B_3$ = UGU, UGU codes the protein Cys.
4. UGGUGGUGG: $P_4 = \{\text{ Trp, Gly, Val}\}$, $B_4 = \{\text{UGG, GGU, GUG }\} \Rightarrow$ Since $P_2 \cap P_4 =$ Val and $B_2 \cap B_4 =$ GUG, we know that GUG is coding for Val.

⋮

4

RNA

4.1 Role of RNA

RNA plays an important role in our live as it participate in the fundamental functions.We have to differentiate between coding and non-coding RNA. mRNA is a coding RNA since in the protein biosynthesis mRNA (messenger RNA) is first transcribed from DNA and then carries the coding information from the nulceus to the ribosomes. Moreover there are several non-coding RNA (ncRNA) types, such as tRNA (transfer RNA), rRNA (ribosomal RNA) and snRNA(small nuclear RNA, important for RNA splicing).

4.2 The RNA-World-Hypothesis

The RNA world hypothesis proposes that life based on ribonucleic acid (RNA) pre-dates the current world of life based on deoxyribonucleic acid (DNA), RNA and proteins. RNA is able both to store genetic information, like DNA, and to catalyze chemical reactions, like an enzyme protein. It may therefore have supported pre-cellular life and been a major step in the evolution of cellular life.

4.3 RNA Structures

Definition 4.4 (Base pairing rules)**.** Base pairings must be elements of

$$\mathbb{B} = \{AU, UA, GC, CG, GU, UG\}$$

4.4.1 RNA Primary Structure

Definition 4.5. A RNA sequence or primary structure of length n is considered as a word s, of the alphabet $\mathbb{A} = \{A,C,G,U\}^n$, i.e.

$$s = s_1, \ldots, s_n \in \mathbb{A} = \{A,C,G,U\}^n$$

4.5.1 RNA Secondary Structure

Definition 4.6. Let $s = s_1 \ldots s_n \in \mathbb{A}^n$ and $\theta \in \mathbb{N}$ a fixed parameter.
A secondary structure *Sec* is a collection of ordered pairs (i, j), where $1 \leq i < j \leq n$, s.t. the following properties hold:

1. If $(i, j) \in Sec$, then $s_i s_j \in \mathscr{B} = \{AU, UA, GC, CG, GU, UG\}$.
2. If $(i, j), (k, l) \in Sec$, then it is not the case that $i < k < j < l$.
3. If $(i, j), (k, l) \in Sec$ and $i \in (k, l)$ implies that $i = k$ and $j = l$.
4. If $(i, j) \in Sec$, then $j > i + \theta$, where θ is fixed and usually taken to be 3.

If we are given an arbitrary secondary structure *Sec* without having the sequence s, just ignore (1).

4.6.1 Realisation of Secondary Structures

A sequence $s \in \mathbb{A} = \{A,C,G,U\}^n$ realizes a secondary structure *Sec* if $\forall (i, j) \in Sec \colon s_i s_j \in \mathbb{B}$.

4.6.2 Representation of RNA Secondary Structures

There are several possibilities to display RNA structures.

1. The Graph Representation:

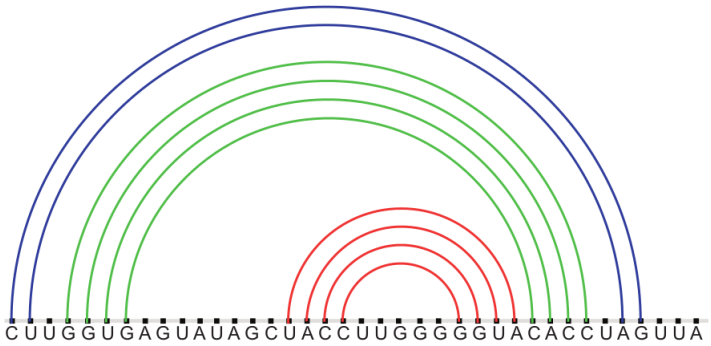


Figure 4.1: Graph representation of a secondary structure

2. The Circle Plot or Nussinov Plot
   Each base is represented as a dot on the circle, progressively from 1 through n. For bases s_i, s_j that are bonded together a line is drawn between i and j.

3. Bracket Representation
   Given an RNA secondary structure *Sec* represented by a set of ordered pairs (i, j) representing the base pairs. We retrieve the bracket representation for each base s_i in *Sec* by the following rule:

$$s_i = \begin{cases} (& \text{if } (i,j) \in Sec \\ ) & \text{if } (j,i) \in Sec \\ . & \text{otherwise} \end{cases}$$

4.6.3 Combinatorics

What is the number of possible secondary structures between i and j?

$$N_{i,j} = N_{i+1,j} + \sum_{\substack{k \\ (i,k) \in Sec}} N_{i-1,k-1} \cdot N_{k+1,j}$$

Lemma 4.1. *Let* $S(n) = N_{1,n}, S(0) = 0, S(1) = 1$. *For* $n \geq 2$ *holds:*

$$S(n+1) = S(n) + S(n-1) + \sum_{k=2}^{n-1} S(k-1)S(n-k)$$

Proof. $S(2) = 1$ $S(l)$ true $\forall l \leq n$ We consider arbitrary length $l \leq n$. If element $n+1$ is not paired

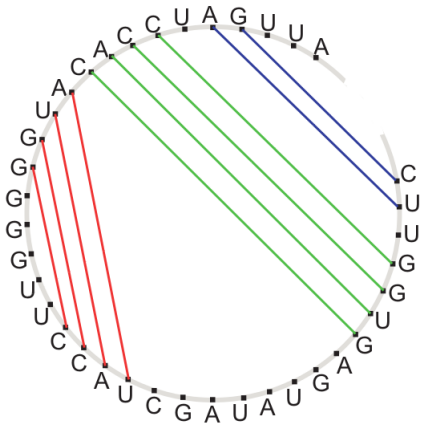


Figure 4.2: Circle plot representation of a secondary structure

with any other vertex, we just have $S(n)$ secondary structures. However, if $n+1$ is paired with some other element $k < n$ we have to consider all possible structures $S(n-1)$ and, in addition, all possible combinations of structures that can be generated by pairing $n+1$ with an element k (all substructures left of k and right of k, i.e. from 1 to $k-1$ and from 1 to $n-k$). □

Lemma 4.2. *For* $n \geq 1 : S(n) \geq 2^{n-2}$

Proof. See exercise. □

4.7 Folding Algorithms

4.7.1 Nussinov-Algorithm

The Nussinov algorithm is based on finding the secondary structure of a primary structure $s = (s_1, \ldots, s_n)$ such that this structure has a maximal number of hydrogen bonds because such a structure should minimize the free energy and therefore be the biologically relevant structure. To do this, we set

$$\delta_{i,j} = \begin{cases} 1 & \text{if } s_i, s_j \in B \\ 0 & \text{else} \end{cases}$$

and let $E_{i,j}$ (a matrix or 2D-array) denote the maximal number of base pairs in a secondary structure for s ranging from bases i to j.

Theorem 4.3. *The following recursion holds with $E_{i,j} = 0$ if $|i-j| \leq 1$ for $\Theta = 1$:*

$$E_{i,j} = \max\{E_{i+1,j}, \max_{i \leq k \leq j}\{(E_{i+1,k-1} + E_{k+1,j+1})\delta_{i,k}\}$$

$E_{i,j}$ can be obtained by analyzing substructures ranging from bases $i+1$ to j and adding individual bases. There are four cases that are to be distinguished.

1. *Add base i such that i and j are paired*
   Look for best substructure for $[i+1, j-1]$
2. *Add base i such that i is unpaired*
   Look for best substructure for $[i+1, j]$
3. *Add node i such that j is unpaired*
   Look for best substructure for $[i, j-1]$
4. *Optimize two substructures $[i, \ldots, k-1]$ and $[k+1, \ldots, j]$*

E can be implemented as an $n \times n$ matrix, where n is the length of the RNA sequence. We inialize the matrix by setting $E_{i,i} = 0$ for $i = 1, \ldots, n$ (diagonal elements) and $E_{i,i-1} = 0$ for $i = 2, \ldots, n$. The final algorithm is given in algorithm 2. To compute the matrix E in the algorithm the following

Algorithm 2 *nussinov*

Require: empty matrix E of size $n \times n$, input sequence s
Ensure: filled matrix E
  for $l \leftarrow 1, \ldots, n$ **do**
    for $i \leftarrow 1, \ldots, n+1-l$ **do**
      $j \leftarrow i + l$
      compute $E_{i,j}$
    end for
  end for

computation is performed:

$$E_{i,j} = \max \begin{cases} E_{i+1,j} & \text{condition 2} \\ E_{i,j-1} & \text{condition 3} \\ E_{i+1,j-1} + \delta_{ij} & \text{condition 1} \\ \max_{i<k<j}\{E_{i,k} + E_{k+1,j}\} & \text{condition 4} \end{cases}$$

Algorithm 3 *nuss − trace*

Require: filled matrix E

Ensure: optimal secondary structure *Sec*

```
push(1, n) on empty stack
while stack is non-empty do
  if i < j then
    if E_{i,j} = E_{i+1,j} then
      push(i+1, j)
    else if E_{i,j} = E_{i,j-1} then
      push(i, j-1)
    else if E_{i,j} = E_{i+1,j-1} + δ_{i,j} then
      push(i+1, j-1)
    else
      for k = i+1, ..., j do
        if E_{i,j} = E_{i,k} + E_{k+1,j} then
          push(i, k)
          push(k+1, j)
        end if
      end for
    end if
  end if
end while
```

To determine the optimal (maximal number of bonds) secondary structure, we can just traceback the matrix using the procedure given in algorithm 3. To receive a better result, one can determine the state of minimal free energy ($\Delta G = \Delta H - T\Delta S$) more explicitly by setting bonding energies in $kcal/mol$ to -3 for CG, -2 for AU, and -1 for GU. This can be incorporated by setting $d_{i,j}$ to these values if i and j fulfill the requirements. This gives a model whose energy can be characterized experimentally.

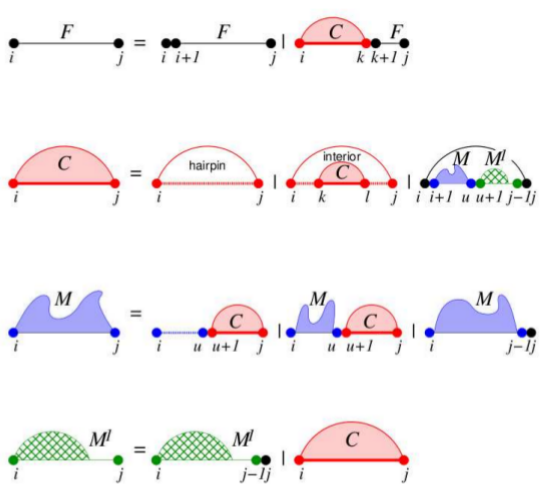
4.7.2 Structural Elements

- i: unpaired
- Base pair (i, j) closing a hairpin-loop: we have $i < k < j$ for all inner bases k
- Base pairs (i, j) and (i', j') for internal loop if we have $i < i' < j < j'$, $i = i+1$ with $j' = j-1$,

and unpaired interior bases

- Bulge loop: (i,j) and $(i',j-1)$ or vice versa
- k-multiloops: $(i,j)[i+1,i'-1](i',j')[i'+1,j'-1].....[j^k+1,j-1]$

4.7.3 MFE-Folding Algorithm



$$F_{i,j} = \min\left\{\begin{array}{l} F_{i+1,j}, \\ \min\limits_{i<k\leq j} C_{i,k} + F_{k+1,j} \end{array}\right\}$$

$$C_{i,j} = \min\left\{\begin{array}{l} \mathcal{H}(i,j), \\ \min\limits_{i<k<l<j} C_{k,l} + \mathcal{I}(i,j;k,l), \\ \min\limits_{i<u<j} M_{i+1,u} + M^1_{u+1,j-1} + a \end{array}\right\}$$

$$M_{i,j} = \min\left\{\begin{array}{l} \min\limits_{i<u<j} (u-i+1)c + C_{u+1,j} + b, \\ \min\limits_{i<u<j} M_{i,u} + C_{u+1,j} + b, \\ M_{i,j-1} + c \end{array}\right\}$$

$$M^1_{i,j} = \min\left\{\begin{array}{l} M^1_{i,j-1} + c, \\ C_{i,j} + b \end{array}\right\}$$

Figure 4.3: The four cases for the MFE-Folding Algorithm by Zucker.

Initialize $F_{ii} = 0$. $C_{ij} = M_{ij} = M^{\infty}_{ij} = \infty$. Compute F_{1n} recursively like `Nussinov`. In figure 4.3 the different recursion cases are shown.

4.8 Realizability

Definition 4.9. A sequence $s \in \mathbb{A} = \{A,C,G,U\}^n$ realizes a secondary structure S if for all $(b_i,b_j) \in S$ holds $(s_i,s_j) \in \mathbb{B}$. Let $C(S)$ denote the set of all sequences realizing the secondary structure S. The size of a secondary structure is denoted as the number of nucleotides.

Definition 4.10. Given secondary structures $S_1,\ldots,S_k$. A shape graph is a graph $G(S_1,\ldots,S_k) = (V,\bigcup S_i)$ such that G contains each edge of all the input structures.

Theorem 4.4. *Given secondary structure S,S' of same size.*

$$C(S) \cap C(S') \neq \emptyset$$

Proof.
Let S, S' be as given, and $G(S,S')$. We construct a sequence $s \in C(S) \cap C(S')$. Each vertex is incident in the shape graph to at most two edges. The connected components of $G(S,S')$ can only be paths or cycles. There are three possible types of position:

1. isolated vertices $\rightarrow$ arbitrary letters
2. positions (i,j) that are paired both in S and S'm this means the form a single edge in the shape graph. We can use any $s_i s_j \in \mathbb{B}$
3. position i is paired differentially in S and S'.

□

Theorem 4.5. *Given secondary structures $S_1, \ldots, S_k$ of size n, the following assumption holds:*

$$\bigcap_{i=1}^{k} C(S_i) \neq \emptyset \Leftrightarrow \Phi = G(S_1, \ldots, S_k) \text{ is bipartite}$$

Proof. Let $\Phi = G(S_1, \ldots, S_k) = (V,E)$ be bipartite, then V can be devided into the two bipartitions $V = V_1 \dot{\cup} V_2$ Then we can assign to each $s_i \colon v_i \in V_1$ an x and to each $s_j \colon v_j \in V_2$ an y s.t. $xy \in \mathbb{B}$.

Let $\bigcap_{i=1}^{k} C(S_i) \neq \emptyset$.Let $\Phi = G(S_1, \ldots, S_k) = (V,E)$ be not bipartite. Then there exists at least one odd cycle C in this graph. According to the base pairing rules $\mathbb{B}$ there is always an even number of steps after a base can encounter again. Thus if we start with an arbitrary letter $x \in \{A,C,G,U\}$ then all the other occurences of x must appear after an even number of steps along this cycle C.↯ □

Definition 4.11. A sequence $s \in \{0,1\}^n$ realizes a secondary structure $S \Leftrightarrow \forall (i,j) \in S \colon s_i \neq s_j$

Corollary 4.6.

$$\bigcup_{i=1}^{k} C(S_i) \neq \emptyset \Leftrightarrow \exists \text{ single sequence } s \in \{0,1\}^n \text{ realizing all } S_i$$

Problem 1. Given a sequence $s \in \{0,1,\star\}^n$, where $\star$ is a "don't care symbol". The sequence s realizes a secondary structure S if $\forall (i,j) \in S \colon s_i \neq s_j$ or $s_i = s_j = \star$

4.11.1 Min⋆Realization Problem [M⋆RP]

Given secondary structures $S_1, \ldots, S_k$. Find a single sequence $s \in \{0,1,\star\}^n$ realizing $S_1, \ldots, S_k$ s.t. the number of $\star$ is minimal.

4.11.2 Algorithm `minV_bip(H)`

Given a graph $H = (V,E)$. remove the number of vertices of H s.t. H becomes bipartite.

$Min \star RP \Leftrightarrow$ `minV_bip` $(G(S_1,\ldots,S_k))$

Definition 4.12. Given a graph $G = (V,E)$. Let $V' \subseteq V$. V' is a Feedback Vertex Set for odd Cycles (FVOC) if it contains at least one vertex from every odd cycle.

Problem 2. Given a graph $G = (V,E)$ and an integer k. Let $\Delta(G)$ be fixed. Is there a $FVOC$ of G of size k. This decision problem is equivalent to the problem `minV_bip(G)`

Theorem 4.7. *`minV_bip(G)` is NP-complete for* $\Delta(G) \geq 4$.

Proof.

1. `minV_bip(G)` $\in NP$
   Given a solution which is by construction a $FVOC$, we remove this set of vertices and obtain then G'. With `BFS` we check if G' is bipartite.
2. `minV_bip(G)` is NP-hard
   We will show it by a polynomial reduction from the Vertex Cover problem to the `minV_bip(G)` problem $VC \leq_p$ `minV_bip(G)` Let $G = (V,E)$ and $\Delta(G) = 3$. Construct a new graph $\tilde{G} = (\tilde{V},\tilde{E})$ in the following way:
   $\tilde{V}$ consists of

   a) V the vertex set of G

   b) for each $v \in V$ we add a new vertex $v' \in V'$

   c) for each edge $e \in E$ we add a new vertex $v_e \in V''$

   d) $\tilde{V} = V \cup V' \cup V''$

   $\tilde{E}$ consist of

   a) E the edge set of G

   b) fo each edge $e = (a,b) \in E$ add two new edges (a', v_e) and (v_e, b') in the edge set E'

   c) $\forall v \in V$: add $(v,v') \in E''$

d) $\tilde{E} = E \cup E' \cup E''$

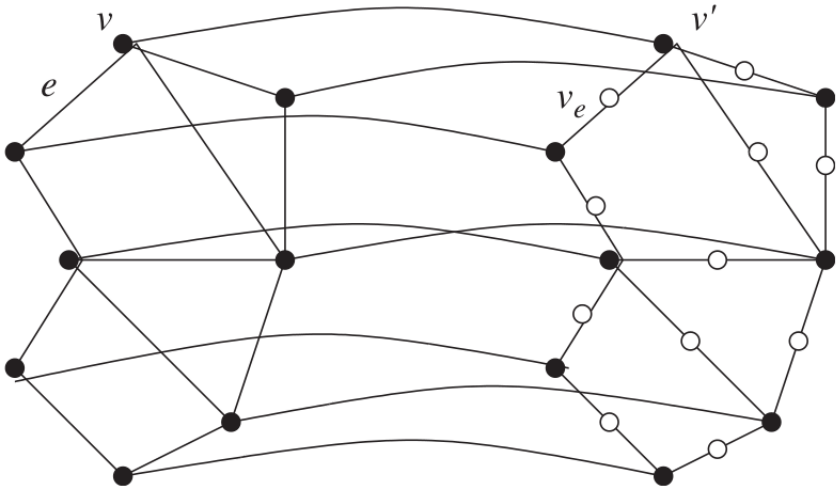


Figure 4.4: new graph $\tilde{G}$

Obiously, this construction can be done in polynomial time.
Let the vertex cover VC of G be of size m. This is equivalent to the statement that $FVOC$ of $\tilde{G}$ has size m. Let be $G, VC(G)$ given and C be an odd cycle in $\tilde{G}$. C cannot be entirely contained in the subgraph $|((V' \cup V''), E')| \leq |\tilde{G}|$, this implies that C contains at least one edge of E. Consequently, C has to contain at least one vertex $u \in VC(G)$. Hence, the vertex cover $VC(G)$ is $FVOC$ of $\tilde{G}$ and $|FVOC| \leq |VC(G)|$. Let U' be $FVOC$ of $|\tilde{G}|$ with $|U'| \leq |VC(G)|$. If U' contains a vertex $v_e \in V''$, v_e can be replaced either by a' or b' and U' is still a $FVOC$. Thus without loss of generalization $|U'| \leq |V \cup V'|$. Let $|U''| \leq |V|$ s.t. U'' consists of all vertices $v \in V : v \in U' \vee v' \in U'$. In total we have now a construction s.t. $|U''| \leq |U'| \leq |VC(G)|$. Since by construction $|U''| \leq |V|$, U'' cannot be a VC of G. Thus there exists an edge $e = (v_1, v_2) \in E$ that is not covered by U'', this means neither v_1 nor v_2 is contained in U''. Consequently, $v_1, v_1', v_2, v_2' \notin U'$. This contradicts the assumption that U' is a $FVOC$.

□

4.12.1 Odd Homeomorphic Extension

Motivation

Given $G(S_1, \ldots, S_k)$. Is the problem `minV_bip` $(G(S_1, \ldots, S_k))$ easier?

Example. Given graph G. How many secondary substructures $S_1,\ldots,S_k$ do we need in order that $G(S_1,\ldots,S_k) \cong G$.

Definition 4.13. Given a graph $G=(V,E)$ then $G'=(V',E')$ is an odd homeomorphic extension of G, if G' can be obtained from G by replacing edges $e=(v_1,v_2)$ of G by paths of odd length of the form $(v_1,u_1,\ldots,u_k,v_2)$ where $deg(u_i)=2 \wedge u_i \notin V$.

Theorem 4.8. *Let $G=(V,E)$ be given and $\Delta(G)=k\geq 3$. Then there exists at least $S_1,\ldots,S_k$ s.t. $G(S_1,\ldots,S_k)\cong G'$ where G' is an odd homeomorphic extension of G. In particular $S_1,\ldots,S_k$ can be constructed in O($|E|$) time.*

Proof. omitted. See for reference *Journal of theoretical biology 2005 p:216-227, On realizing shapes in theory of RNA neutral networks* □

Example.

Since $G\cong\tilde{G}\cong G'$ and $VC\leq_p$ `minV_bip(G)` `minV_bip(G(` $S_1,\ldots,S_k$`))` is NP-comple, too.

Lemma 4.9. *FVOC for odd cycles in G is FVOC in G' and vice versa.*

Corollary 4.10. `minV_bip(G(` $S_1,\ldots,S_k$`))` *is NP-complete for $k\geq 4$ which is equivalent to M$\star$ RP is NP-complete for $k\geq 4$.*

4.14 Inverse Folding Algorithms

Problem 3. Given a secondary structure S. Find a sequence s s.t. $s \xrightarrow{mfe} S$ where $s\in C(S)$.
The problem to find inverse folding algorithms is e... by the fact that not to all secondary structures S there exist a sequence s s.t. $s \xrightarrow{mfe} S$.

Problem 4. For a given secondary structure S find sequence s s.t. $s \xrightarrow{mfe} S' \wedge d(S,S')$ is minimal. The distance of two secondary structures can for example be computed as the symmetric union of both structures:

$$d(S,S') = S \triangle S' = S \cup S' \setminus (S \cap S')$$

5

Product Graphs

5.1 Motivation

In the study of inheritance of traits and their evolution, one have to distinct between genotype and pheotype of an organism. The phenotype can be defined as a set of all characters with its characteristics (traits) of an organism. The question is how the characterisation of a character is determined. One of the results was that characters can vary independently. This however is equivalent to (local) prime factors of the phenotype space.

5.2 Properties

1. The product of a simple graph is a simple graph. This means that following properties hold:

    - The product graph has no loops,
    - no multiple edges,

- no hyperedges.

2. The vertex set of a product is the Cartesian product of the vertex sets of the factors.
3. The adjacency in the product depends on adjacency properties of the projections of pairs of vertices into factors.

There are 256 possibilities to define such a graph product, but only six of them are commutative, associative and have a unit. If one wishes the product to depend on the structure of both factors and if the homomorphism property of the projections into the factors, that will be defined later on, plays a role, the number of products decreases to 4. In this contribution we are concerned with two of these 4 products, the Cartesian and the strong product. In particular, we are interested in the strong product, but as it turns out the Cartesian product is closely related to the strong product and plays a central role in the prime factorization of strong product graphs. Consequently, we will also deal with the Cartesian product.

5.3 Standard Products

There are 4 standard products:

1. Cartesian Product $G\Box H$
   The vertex set of the Cartesian product $G_1\Box G_2$ of two graphs G_1 and G_2 is the set

   $$V(G)\times V(H) = \{(v_1,v_2)|v_1 \in V(G), v_2 \in V(H)\},$$

   that is, the Cartesian product of the vertex sets of the factors.
   Two vertices $(x_1,x_2),(y_1,y_2)$ are adjacent in the Cartesian product $G_1\Box G_2$ if one of the following conditions is satisfied:

   (i) $(x_1,y_1) \in E(G_1)$ and $x_2 = y_2$

   (ii) $(x_2,y_2) \in E(G_2)$ and $x_1 = y_1$

2. Direct Product $G\times H$
   Two vertices $(x_1,x_2),(y_1,y_2)$ are adjacent in the direct product $G_1 \times G_2$ if the following condition is fulfilled:

$$(x_1, y_1) \in E(G_1) \text{ and } (x_2, y_2) \in E(G_2)$$

3. Strong Product $G \boxtimes H$

   The Strong product is a union of the Cartesian and the Direct product, s.t.

$$E(G_1 \boxtimes G_2) = E(G_1 \square G_2) \cup E(G_1 \times G_2)$$

4. Lexicograph product $G \circ H$

   Two vertices $(x_1, x_2), (y_1, y_2)$ are adjacent in the Lexicograph product $G_1 \circ G_2$ if one of the following conditions is satisfied:

   (i) $(x_1, y_1) \in E(G_1)$

   (ii) $(x_2, y_2) \in E(G_2)$ and $x_1 = y_1$

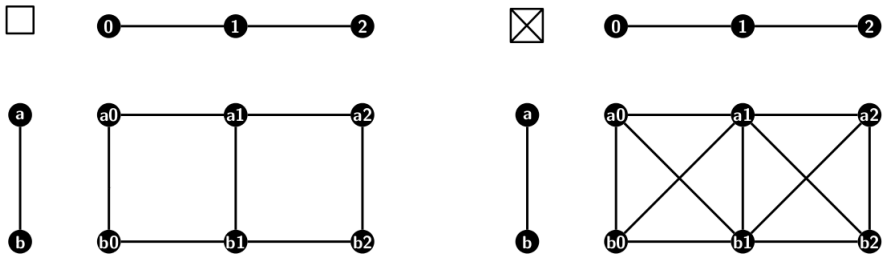


Figure 5.1: Left: Cartesian product Right: Strong product

5.3.1 Cartesian Graph Products

When using the Cartesian graph product, the multiplication of paths will always result in grids. We define the hypercube with n dimensions as

$$Q_n = \square_{i=1}^{n} K_2$$

The copies of factors are so called layer or fibers.

Definition 5.4 (Layer of Fiber)**.** A layer H_i^w is defined as

$$H_i^w = \langle \{v \in V | v_i \neq w_i, w_j = v_j, j \neq i\} \cup \{w\} \rangle$$

where $\langle\rangle$ indicates an induced subgraph and H is given by

$$H = \star_{i=1}^{n} H_i = (V, E)$$

Definition 5.5 (Homomorphism)**.** A homomorphism is defined by a function $\phi : V(G) \to V(H)$ such that $(u,v) \in E(G) \Rightarrow (\phi(u), \phi(v)) \in E(H)$.

Definition 5.6 (Isomorphism)**.** An isomorphism is defined by a function $\phi : V(G) \to V(H)$ such that $(u,v) \in E(G) \Leftrightarrow (\phi(u), \phi(v)) \in E(H)$. It is a stronger form of homomorphism because equivalence, in contrary to implication, has to hold.

Definition 5.7 (Weak Homomorphism)**.** Given two graphs G, H. There is a week homomorphism between G and H if the following condition is satisfied:

$$(u,v) \in E(G) \Rightarrow (\phi(u), \phi(v)) \in E(H) \vee \phi(u) = \phi(v),$$

that means that edges from G are mapped either to edges of H or to vertices of H.

Definition 5.8 (Projection)**.** The projection of $G \square H$ into G is defined by the mapping function

$$P_G \colon G \square H \mapsto G, (g,h) \mapsto g$$

Analogously,the projection of $G \square H$ into H is given by

$$P_H \colon G \square H \mapsto H, (g,h) \mapsto h$$

A projection is a week homomorphism.

Properties

The Cartesian product has the following properties:

1. Commutativity
$$G \square H \cong H \square G$$
2. Assocciativity
$$(G_1 \square G_2) \square G_3 \cong G_1 \square (G_2 \square G_3)$$
Thus we can write the product graph $G = G_1 \square G_2 \square \ldots \square G_n = \square_{i=1}^{n} G_i$.

3. The one-vertex complete graph K_1 serves as a unit element for Cartesian products, since

$$K_1 \square G_1 \cong G_1 \square K_1 \cong G_1$$

4. Distributivity (using disjoint union '+')

$$G_1 \square (H_1 + H_2) = (G_1 \square H_1) + (G_1 \square H_2)$$

5.8.1 Distances

Theorem 5.1. *Let* $((g,h),(g',h')) \in V(G \square H)$.

$$d_{G \square H}((g,h),(g',h')) = d_G(g,g') + d_H(h,h')$$

Proof.

Case 1: Without loss of generation we will perform this case for $d_G(g,g')$, it works analogous for $d(h,h')$. $d_G(g,g') = \infty \Rightarrow \exists$ no path between g and g' in G. Thus G is not connected and obviously, we can denote G as the disjoint union of G_1 and G_2. Consequently the product graph will again consist of two disjoint unions:

$$G \square H = (G_1 + G_2) \square H = G_1 \square H + G_2 \square H \Rightarrow d_{G \square H}((g,h),(g',h')) = \infty$$

Case 2:

(1) Let $d_G(g,g'), d_H(h,h') < \infty$. Then there exists a path $P_G = (g,\ldots,g')$ in G and $P_H = (h,\ldots,h')$ in H, respectively. Let denote the path $P_{G \square H} = ((g,h),\ldots,(g',h'))$ as R. Thus the number of edges in R can be computed by the sum of the two distances $d_G(g,g')$ and $d_H(h,h')$ and we can estimate the distance $d_{G \square H}((g,h),(g',h'))$:

$$d_{G \square H}((g,h),(g',h')) \leq d_G(g,g') + d_H(h,h')$$

(2) Let another path R' be a shortest path in $G \square H$ between (g,h) and (g',h'). Since projections are week homomorphisms, the edges of R' are mapped to edges in H or vertices in G via a projection P_H or vice versa.

$$d_{G \square H}((g,h),(g',h')) = d_G|E(R)| = |E_{P_G}(R')| + |E_{P_H}(R')| \geq d_G(g,g') + d_H(h,h')$$

Conclusion

$$d_{G\square H}((g,h),(g',h')) = d_G(g,g') + d_H(h,h')$$

□

Corollary 5.2. *Let $G = \square_{i=1}^{n} G_i$. For any two vertices $x = (x_1,\ldots,x_n)$, $y = (y_1,\ldots,y_n) \in V(G)$*

$$d_{G\square H}(x,y) = \sum_{i=1}^{n} d_{G_i}(x_i,y_i)$$

Corollary 5.3. *$G = \square_{i=1}^{n} G_i$ is connected if and only if G_i is connected $\forall i \in \{1,\ldots,n\}$.*
This means that G is connected if and only if each factor in G is connected.

5.9 Prime Factor Decomposition (PFD)

Definition 5.10. A graph G is considered prime, when every representation of G as $G_1 \star G_2$, where $\star$ denotes some graph product, implicates that we have $G_1 \simeq K_1 \vee G_2 \simeq K_1$. This means that one of the graphs has to be isomorphic to K_1, which is considered a unit element under the graph product $\star$ if we have $G \simeq G \star K_1$.

Lemma 5.4. *Let e and f be two incident edges in G (under the Cartesian products), stemming from different fibers (layers). Then there exists exactly one ($\exists!$) diagonal-free square containing both e and f.*

Proof. See exercise. □

Definition 5.11 (Diagonal Free Square)**.** A diagonal free square is a cycle $C_n = v_1,\ldots,v_n,v_1$ that does not contain any other edges than that of the cycle.

5.11.1 Nomenclature

The following nomenclature is used regarding graphs:

- K_n: complete graph with n vertices
- P_n: path with n vertices
- C_n: cycle with n vertices

Theorem 5.5 (Unique PFD for Cartesian Product Graphs)**.** *Every connected graph $G = (V,E)$ has a unique representation as a Cartesian product of prime factors (up to isomorphism and the order of factors). The number of prime factors is at most $log_2(n)$, where $n = |V|$. We have $O(2^P) = O(n)$ with $p \leq log_2(n)$.*
PFD is not unique in the class of disconnected graphs.

Theorem 5.6. *No Unique PFD for connected Direct Product Graphs.*

Proof. Let us have $H_1 \square H_2 \simeq H_3 \square H_4$. Then we have

$$(K_1 + K_2 + K_2^2) \square (K_1 + K_2^3) \simeq (K_1 + K_2^2 + K_2^4) \square (K_1 + K_2)$$

Here, $+$ refers to disjoint union and K_i^j is given by $\square_{j=1}^{n} K_i$.
We have $H_i = K_1 + A_1 + A_2$ with $A_2 = \emptyset$. Therefore, the number of connected components in a product graph is the same as the product of the number of connected components over all factors.
Let us assume that H_i is not prime for the sake of proving by contradiction. Then we have

$$\begin{aligned} K_1 + A_i + A_2 &\simeq B \square (C_1 + C_2 + C_3) \\ &= B \square C_1 + B \square C_2 + B \square C_3 \\ &= K_1 \end{aligned}$$

Consequently, B has to be equal to K_1 and therefore H_i has to be prime by definition which is a contradiction to our assumption.

□

5.12 Prime Factor Decomposition with respect to □

5.12.1 The Relation σ

Let $G = \square_{i=1}^{n} G_i$ be a Cartesian product graph. Two edges e, f are in relation σ, $(e \sigma f)$, if the endpoints of e, resp. f , differ exactly in the same coordinate i.There can only be one such coordinate by definition of the Cartesian product. Thus, for e and f with $e \sigma f$ holds: they are both edges of fibers of factor G_i. The edges e and f can than be colored with color i. The aim is to compute the finest σ.
An example is presented in figure 5.2. Two edges e and f are in relation σ if there exists some i such that $c(e) = c(f) = i$ holds.

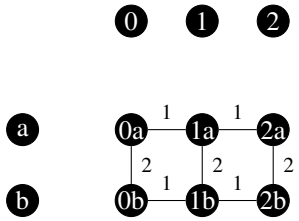


Figure 5.2: Illustration of different values for $c(e)$ given as edge labels. Do note that, here, the second coordinate is considered to the alphabetic characters, even though they denote the row (first coordinate in some sense). By definition of σ we know that all horizontal edges share a σ relation among each other ($c(e) = 1$

and all vertical edges share a σ relation among each other ($c(e) = 2$).

5.12.2 The Djokovic-Winkler-Relation Θ

Two edges $e = (x,y)$, $f = (a,b)$ are in Relation Θ, $(e\Theta f)$, iff

$$d(x,a) + d(y,b) \neq d(x,b) + d(y,a)$$

Let us consider a square with four nodes as an example, see figure 5.3

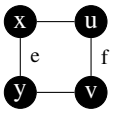


Figure 5.3: Example graph for the Θ relation

Lemma 5.7. *Let G be a graph. It holds:*

(i) Two incident edges e and f be are in relation Θ if and only if they are part of a common triangle.

(ii) Let P be a shortest path in G. Then no two edges of P are in Θ.

(iii) Let C be an isometric cycle of G and $e \neq f$ be edges that are both in C. They are in Θ if and only if e and f are antipodal edges, that is, oppossite of each other.

Θ is symmetric, reflexiv, not transitiv.

The properties of Θ are illustrated in figure 5.4. In this graph, we have $(x,y)\Theta(u,v)$ and $(u,v)\Theta(x,z)$ but there is no $\Theta-$Relation between (x,y) and (x,z).

Definition 5.13 (Convex Graphs)**.** The subgraph $H \subseteq G$ is convex if all shortest paths of G are also shortest paths in H, i.e. H has to contain all shortest paths of the supergraph. Let us consider the example from figure **??**.

Definition 5.14 (Isometric Graphs)**.** The subgraph $H \subseteq G$ is isometric if and only if the distances between all vertices in H are the same as the distances of these vertices in G.

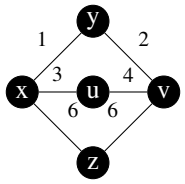


Figure 5.4: Example graph $K_{2,3}$. Edge labels represent an edge enumeration.

We can convert Θ to an equivalence relation in two ways. We can either compute the transitive closure of the graph we deal with or we uniquely enumerate each edge. When referring to Θ in context of a transitivity-closure graph, we write Θ^*. The transitive closure of a graph is the smallest relation containing Θ that is transitive.

Theorem 5.8. *Let us have two edges e and f of a Cartesian product graph with $e\Theta f$, thus $c(e) = c(f) = i$ (coordinates differ at i-th position). We then have*

$$\begin{aligned} e\Theta f &\Rightarrow \exists i : c(e) = c(f) = i \\ &\Rightarrow e\sigma f \\ &\Rightarrow \Theta \subset \sigma \\ &\overset{\text{trans. closure}}{\Rightarrow} \Theta^* \subseteq \sigma \end{aligned}$$

Figure 5.5 shows an example of three equivalence classes and the τ relation.

5.14.1 The Relation τ

We have $e\tau f$ for $e = (u,v)$, $f = (u,w)$ if and only if we have

(i) $e = f$ or

(ii) u is the only common neighbor of v and w, and $(v,w) \notin E(G)$

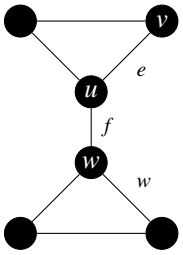


Figure 5.5: We have three equivalence classes (Θ^* (two triangles and the edge f) and $e\tau f$. It is possible to merge equivalent classes (some algorithm for prime graphs?).

$e\tau f$ implicates that e and f are incident and are assigned different colors according to Θ. Also, the square-property (diagonal-free square) is not fulfilled. In addition, the edges must be in a fiber of a single factor.

We have $\tau \subseteq \sigma$. Because we have $\Theta \subseteq \sigma$ and $\sigma^* \subseteq \sigma$ it is implied that $(\Theta \cup \tau)^* \subseteq \sigma$. We define $\Pi = (\sigma \cup \tau)^*$. The relation $(\Theta \cup \tau)^*$ is the finest product relation σ and thus corresponds to the PFD w.r.t. $\square$ of a given graph.

Lemma 5.9. Π *is contained in any product relation* σ *and satisfies* f *the square-property.*

Proof. We have $\Pi = (\Theta \cup \tau)^* \subseteq \sigma$. Let us furthermore have e and f with $e = (u,v)$ and $f = (u,w)$ This implies that e and f are incident to each other, as illustrated in figure 5.6. We can distinguish two cases. We have a triangle u,v,w,u. Then we have $e\Theta f$. This, however, contradicts that we have different Π-classes (different factors). In the second case, we do not have a triangle or a square and therefofre have $e\tau f$. This is a contradiction to our assumption. Now let us consider the third case. Here, we have $K \geq 1$ squares.

If we have $K > 1$ squares then we can consider the $K_{2,3}$ subgraph for which $e\Theta^* f$ holds, which is a contradiction. We have exactly one square and exactly one ($\exists!$) diagonal-free square, that is, the square-property is fulfilled. $\square$

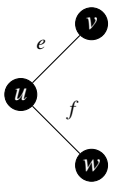


Figure 5.6: Example

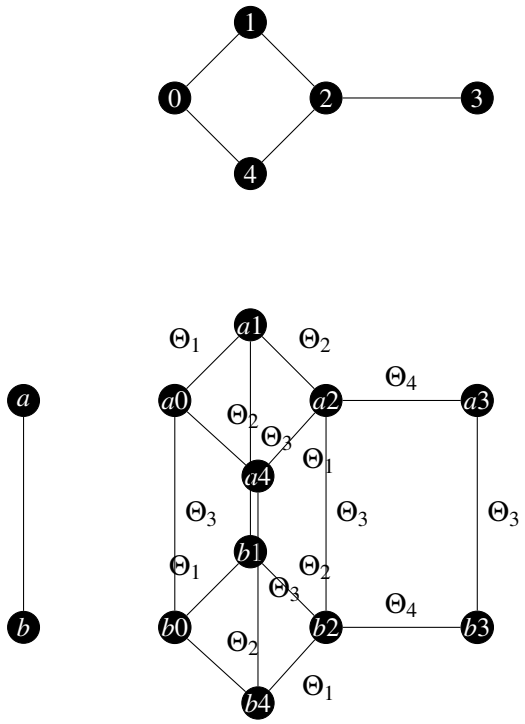


Figure 5.7: Illustration for the PFD-algorithm. We have four Θ classes: Θ_1, Θ_2, Θ_3, and Θ_4.

5.15 Prime Factor Decomposition (PFD) with respect to □

The input of this algorithm is the adjacency list of a connected graph $G = (V, E)$. The output are the prime factors of G with regard to □.

1. Compute equivalence classes of $(\Theta \cup \tau)^*$ (transitive closure), which implies an edge coloring.
2. For $i = 1, \ldots, |\text{equivalence classes of } \Pi|$, color all edges of Π_i (the i-th equivalence class) with color i
3. For $i = 1, \ldots, |\text{equivalence classes of } \Pi_i|$, take out one connected component of each color, store G, the prime factor of G_i

The algorithm is illustrated in figure 5.7. All colors with overlaps with the τ relation can be merged to yield the prime factor decomposition.

Lingo: Reflection Reflecting a graph is basically the same as drawing the graph in a reasonable fashion.

5.16 Prime Factor Decomposition with respect to $\boxtimes$

Given a strong product $G = G_1 \boxtimes \ldots \boxtimes G_n$. Find special subgraphs of G, namely Cartesian Skeleton $S(G)$. Decompose $S(G)$ with respect to $\square$ in $G_1, G, \ldots, G_l$ where $l \geq n$. Additional operation lead to PFD of G.

5.16.1 Problem

The Cartesian Skeleton is not uniquely determined if $\exists v, w \in V(G)\colon\ N[v] = N[w]$ then v, w are in relation S.

Definition 5.17 (Closed Neighbourhood)**.** $N[v] = \{w|(vw) \in E(G)\} \cap \{v\}$

Definition 5.18 (S-thin)**.** A graph G is S-thin if and only if $\forall v, w \in V(G), v \neq wN[v] \neq N[w]$. G/S is thin. Moreover $(G \boxtimes H)/S = G/S + H/S$

Definition 5.19 (Quotient Graph)**.** Given equivalence relation classes S_i with $i = 1, \ldots, l$, we call the Graph G/S a quotient graph.

$$(S_i, S_j) \in E(G/S) \Leftrightarrow \exists u \in S_i, v \in S_j\colon\ uv \in E(G)$$

6

Phylogenetic Reconstructions

6.1 Introduction

- Phylogenetics is a study of evolutionary history of genes, species or other taxa.
- The aim is to assemble a phylogenetic tree that represents a hypothesis about the evolutionary ancestor of genes, species and other taxa.

6.2 Methods

There are several methods for creating a tree of taxa representing the evolutionary history. There can be devided into two groups:

Distance based methods

Here, the distance between two species reflects the period of time in the past when the species diverged. The aim is to find a tree that is consistent with the data/distance matrix.

Character based methods

The second method is based on characters, which can be binary characters (i.e Has tail or not), qualitative characters i.e. number of limbs, vertebras and character strings representing f.ex. genes. We differentiate between three different character based methods:

1. **Parsimony Method**
   The idea is that evolution is lazy and produce a minimal number on events(mutations). The aim is to find a tree according to some model of evolutionary change (?) that explains the data best.
2. **Probabilistic Method**
   Probabilistic methods like maximum Likelihood method is used to describe evolution. Evolutionary changes are assumed to appear with a particular probability. Again we are looking for a tree taht fits best to the model.
3. **Consensus Method**
   Find a supertree or greates common subtree for gene trees that reflects each of the gene trees best.

6.3 Distance Based Method

For given distances between species we want to compute trees.

Definition 6.4. A proper distance matrix D is a symmetric $n \times n$ matrix such that

$$D(i,i) = 0 \wedge D_{ij} > 0, i \neq j$$

Definition 6.5 (Metric)**.** A distance matrix D induces a metric if in addition the triangle inequality holds

$$\forall i,j,k \in \{1,\ldots,n\}\, D_{ij} \leq D_{ik} + D_{kj}$$

The question is when a metric space (X,D) has a tree topology. A metric space can induce an ultrametric tree or an additive tree. The idea of a constant molecular clock, i.e. that each evolutionary change/mutation appears with the same probability at each time, location etc. is the interpretation of ultrametric trees. This kind of tree is rooted, i.e. we have a direction from root to the leaves. The different species are represented by the leaves, and the aim is to find the right topology of the tree with respect to the leaves.

An additive tree is an unrooted tree with bijective labeling $1,\ldots,n$ genes to the leaves. Moreover an arbitrary tree T is additive if $D_{ij}(T) = d(i,j)_T$, thus we have edge weights.

Question: Given a distance matrix D,when can we find a corresponding tree?

Theorem 6.1. *D induces an ultrametric if and only if D induces a matrix and in addition holds $\forall i,j,k\colon D_{ij} \leq \max\{D_{ik},D_{kj}\}$*

Lemma 6.2 (Three Point Condition (TPC))**.** *D is an ultrametric on a set X if and only if $\forall i,j,k$ the largest values of D_{ij},D_{ik},D_{kj} are equal.*

Definition 6.6 (Ultrametric Tree)**.** An ultrametric tree T from a distance matrix D has the following properties.

1. T has n leaves labeled bijectively by ....
2. T is a binary tree, i.e. every inner vertex has two children.
3. On every path from a leaf to the root the weights of the inner vertices are increasing.
4. For every pair of leaves i,j the lowest common ancestor should be weighted with D_{ij}

Theorem 6.3. *A distance matrix D has an ultrametric tree T if and only if D induces an ultrametric.*

6.6.1 UPGMA

List of Figures