Alignments using Combinatorial Optimization

Sources for the following lectures:

- Althaus, E., Caprara, A., Lenhof, H.-P., Reinert, K.: Multiple Sequence alignment with arbitrary gap costs:Computing an optimal solution using polyhedral combinatorics. Proceedings of the 1st European Conference on Computational Biology (ECCB 2002), pages 4-16, 2002
- Althaus, E., Caprara, A., Lenhof, H.-P., Reinert, K.: A Branch-and-Cut Algorithm for Multiple Sequence alignment: Mathematical Programming 2005
- J. D. Kececioglu, H.-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, M. Vingron: A Polyhedral Approach to Sequence Alignment Problems. Discrete Applied Mathematics, volume 104, pages 143-186, 2000.

Motivation

Multiple alignment and structural multiple alignment are hard combinatorial problems. If one wants to solve them exactly, the usual way is to use dynamic programming combined with clever bounding techniques.

Nevertheless, dynamic programming has clear limitations. If one considers for example the generalization of the pairwise dynamic programming algorithm for multiple alignment, we have to consider $O(2^k)$ choices in *each* step of the algorithm, *and* we have to store intermediate results. This is clearly only feasible for $k \approx 10$ sequences.

Similar statements hold for structural alignments.

Motivation (2)

Why bother? Why do we want to compute *exact* solutions to these problems? Are heuristics not sufficient, especially considering that all scoring schemes only approximate the biological truth?

The answer is twofold:

1. In order to know how well a scoring scheme models the biological truth, we need exact results to compare them to experimentally derived findings. For example we can use the database *BaliBase*, which is based on the alignment of structural protein units to evaluate the performance of a given scoring scheme.
2. If we know that a scoring scheme approximates the problem at hand very well, then it might be worthwile to compute exact solutions if the problem size allows.

Motivation (3)

In the following lectures we introduce a set of different techniques to solve these problems to optimality. These techniques are based on the formulation of the problems as ILPs (*Integer Linear Programs*). This has two advantages:

1. The problems are modified quite easily by adding new variables or contraints.
2. Solving (integer) linear problems is a field where a lot of research has been done, and there are very efficient techniques known to solve ILPs.

Motivation (4)

In the following lectures we will discuss:

- How to model (structural) multiple alignment problems as graph problems, which in turn are easy to translate into ILPs.
- Approaches to solve the ILPs (e.g., the branch-and-cut approach).
- The most important points one has to address in solving a problem with branch-and-cut:
  - identifying classes of facet-defining inequalities
  - how to solve the associated separation problems

Linear programming

We first give a short overview of linear programming and how to solve (integer) linear programs using software packages.

A *linear program (LP)* consists of a set of linear inequalities,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots a_{1n}x_n &\le b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots a_{2n}x_n &\le b_2 \\ &\dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots a_{mn}x_n &\le b_m, \end{aligned}$$

together with an *objective function*

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

to be optimized, i.e., minimized or maximized.

Linear programming (2)

Linear programs can be efficiently solved using the *simplex method*, developed by George Dantzig in 1947, or using *interior point methods* introduced by Khachiyan in 1979. The simplex algorithm has an exponential worst case complexity but runs quite quickly in practice.

Interior point methods were first only a proof that linear programming can be solved in polynomial time. The original algorithms were not practical. Meanwhile most LP solvers have both methods as practical routines.

Linear programming (3)

There exist powerful computer programs for solving LPs, even when huge numbers of variables and inequalities are involved.

CPLEX is the most well known and powerful commercial LP solver. `SoPlex` is part of the *ZIB Optimization Suite* (`http://zibopt.zib.de`), which is free for academic purposes.

For the rest of the lecture we will consider a LP solver as a black box which can solve linear programs very efficiently and numerically relatively stable.

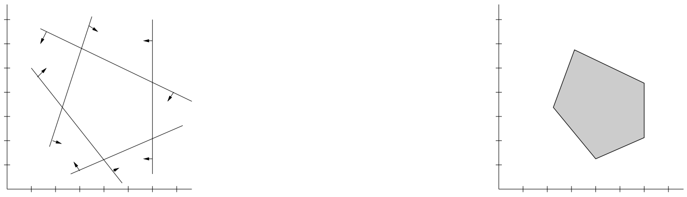
Linear programming (4)

We give a small example of a LP. The inequalities of an LP describe a convex *polyhedron*, which is called a *polytope*, if it is bounded.

For example, the inequalities

$$\begin{aligned} -1x_1 - 1x_2 &\le 5 \\ -2x_1 + 1x_2 &\le -1 \\ 1x_1 + 3x_2 &\le 18 \end{aligned} \qquad \begin{aligned} 1x_1 + 0x_2 &\le 6 \\ 1x_1 - 2x_2 &\le 2 \end{aligned}$$

describe the hyperplanes and polytope depicted in this cartoon:



For example, the objective function $2x_1 - 3x_2$ takes on a maximum of 6, for $x_1 = 6$ and $x_2 = 2$, and a minimum of -9, for $x_1 = 3$ and $x_2 = 5$.

Linear programming (5)

If we are after integral solutions, we were lucky with our example. The solution of the linear program is integer. However, if we change some contraints, we are not so lucky. Assume we enter the following LP into our solver (shown in CPLEX LP format):

```
maximize  2x1-3x2
subject to
 -1x1 -   1x2 <= 5
 -2x1 + 1.5x2 <= -1
  1x1 +   3x2 <= 18
0.8x1 +   0x2 <= 6
  1x1 -   2x2 <= 2
end
```

Then the optimal solution is fractional, namely $x_1 = 7.5$ and $x_2 = 2.75$. However, we often want to constrain the variables to be *integer*, that means we want to solve an ILP.

Integer linear program

An *integer linear program (ILP)* is a linear program with the additional constraint that the variables x_i are only allowed to take on integer values.

Solving ILPs has been shown to be NP-hard. (See the book by Garey and Johnson 1979, for this and many other NP-completeness results.)

There exist a number of different strategies for approximating or solving such an ILP. These strategies usually first attempt to solve *relaxations* of the original problem, which are obtained by dropping some of the inequalities.

A very common relaxation is the *LP-relaxation* of the ILP, which is the LP obtained by dropping the integer condition.

Integer linear program (2)

In the above example we can ask CPLEX to solve an ILP by specifying

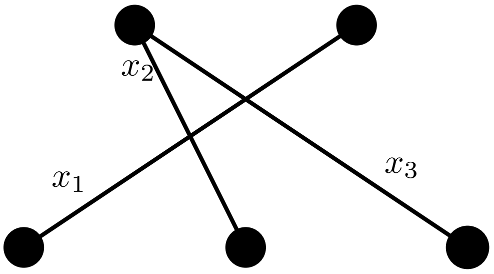
```
general
x1
x2
```

Now the optimal solution is $x_1 = 6$ and $x_2 = 2$.

Among the integer linear programs we have a special class, namely the *combinatorial optimization* problems. In those we restrict the variables to be binary. If we take an “object” into the solution then the associated variable is 1 otherwise it is 0.

Let’s illustrate this on an example. Assume we are given a drawing of a bipartite graph and we want to find the largest subgraph such that no edge crosses or touches another edge in the drawing.

Integer linear program (3)



$$\begin{aligned}&\max x_1 + x_2 + x_3\\&x_1 + x_2 \le 1\\&x_1 + x_3 \le 1\\&x_2 + x_3 \le 1\end{aligned}$$

Obviously we can only choose one of the three edges. However, solving the LP on the right gives us a solution of $x_1 = x_2 = x_3 = \frac{1}{2}$, which satisfies the contraints and maximizes the objective function. If we require the solution to be integer, only one of the three variables is set to 1.

Integer linear program (4)

In CPLEX we can do this by specifying

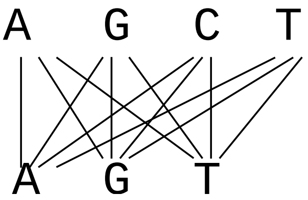
```
binary
x1
x2
x3
```

We now discuss how to model multiple sequence alignment problems as combinatorial optimization problems by first formulating them as a graph problem and then do the obvious 1-to-1 mapping between edges and variables.

The alignment graph

Given two sequences a^1 = `A G C T` and a^2 = `A G T` .

The *complete alignment graph* is the following bipartite graph $G = (V,E)$, with node set V and edge set E:

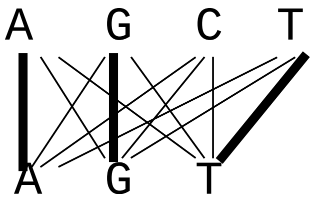


Each edge $e = (u,v)$ has a weight $\omega(e) = s(u,v)$, the score for placing v under u.

An *alignment graph* is any subgraph of the complete alignment graph.

The trace of an alignment

Given an alignment such as $\begin{array}{cccc} \mathtt{A} & \mathtt{G} & \mathtt{C} & \mathtt{T} \\ \mathtt{A} & \mathtt{G} & - & \mathtt{T} \end{array}$, we say that an edge in the alignment graph is *realized*, if the corresponding positions are aligned:



The set of realized edges is called the *trace* of the alignment. An arbitrary subset $T \subseteq E$ of edges is called a *trace*, if there exists some alignment that it realizes.

Similarly, we define the (complete) alignment graph and trace for multiple alignments. For r sequences, the resulting graph will be r-partite.

Maximum-weight trace problem

Problem. Given sequences A and a corresponding alignment graph $G = (V, E)$ with edge weights ω. The maximum-weight trace problem is to find a trace $T \subseteq E$ of maximum weight.

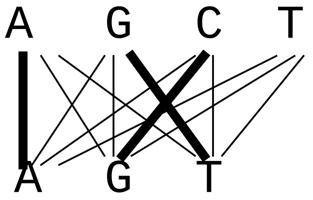
Note, that for two sequences, this can be solved in polynomial time by dynamic programming.

Characterization of traces

We have seen that an alignment can be described by a trace in the complete alignment graph $G = (V, E)$.

Question: Is *every* subset $T \subseteq E$ the trace of some alignment?

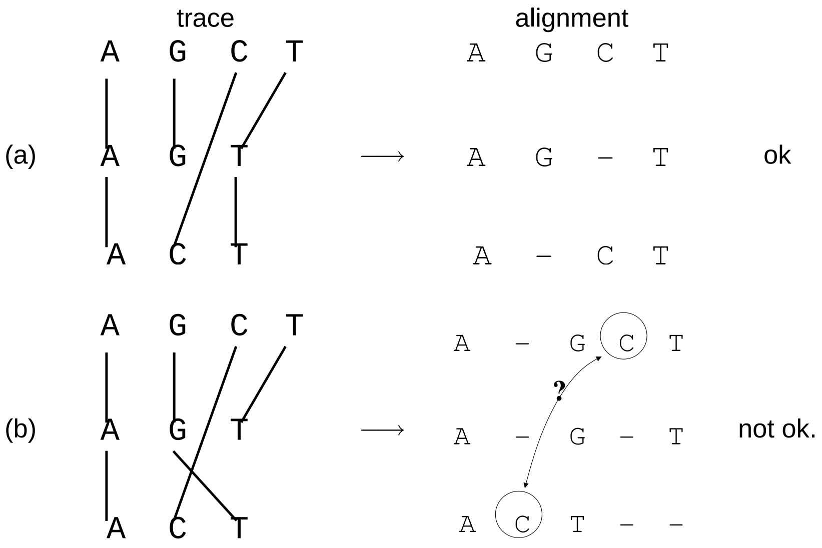
Clearly, the answer is *no*:



Goal: Characterize all legal traces.

Characterization of traces (2)

Here are two examples:



Partial orders

A binary relation $\leq$ is a *(non-strict) partial order*, if it is

1. *reflexive*, i.e., $a \leq a$,
2. *antisymmetric*, i.e., $a \leq b$ and $b \leq a$ implies $a = b$, and
3. *transitive*, i.e., $a \leq b$ and $b \leq c$ implies $a \leq c$.

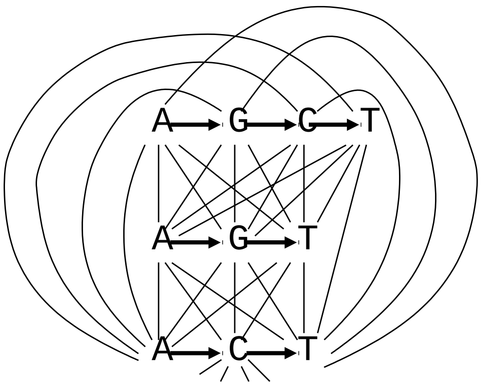
A binary relation $<$ is a *strict partial order*, if it is

1. *irreflexive*, i.e., $a \nless a$, and
2. *transitive*, i.e., $a < b$ and $b < c$ implies $a < c$.

Given a binary relation $\prec$, the *transitive closure* of $\prec$ is a binary relation $\prec^*$ such that $x \prec^* x'$ if there exists a sequence of elements $x = x_1, x_2, \ldots, x_k = x'$ with $x_1 \prec x_2 \prec \ldots x_k$.

The extended alignment graph

We define a binary relation $\prec$ on the characters of the sequences $A = \{a^i_j\}$ by writing $a^i_j \prec a^i_{j'}$, if $j' = j+1$, and indicate the pairs (a^i_j, a^i_{j+1}) by a set H of directed edges in the alignment graph. This results in the *extended* alignment graph $G = (V, E, H)$.



Let $\prec^*$ denote the *transitive closure* of $\prec$, i.e., we write $a^i_j \prec^* a^i_{j'}$, if $j' > j$. Observe that $\prec^*$ is a strict partial order.

The extended alignment graph (2)

Consider two sets of nodes $X \subseteq V$ and $Y \subseteq V$. We define

$$X \lhd Y,$$

if and only if

$$\exists\, x \in X \ \exists\, y \in Y : x \prec y.$$

We define $\lhd^*$ to be the transitive closure of $\lhd$, that is, we write $X \lhd^* Y$, if for one of the sequences $a^p \in A$ we have that X contains a node representing a position a^p_j in a^p and Y contains a node representing another position a^p_k in a^p, with $j < k$.

In other words, we write

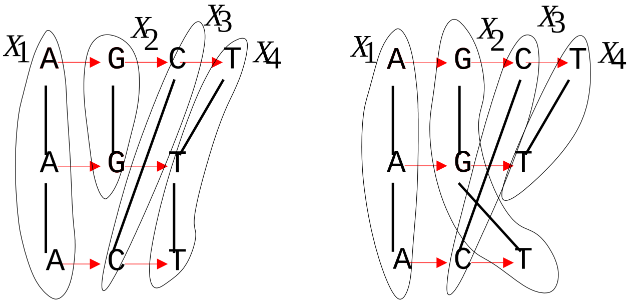
$$X \lhd^* Y,$$

if and only if

$$\exists\, x \in X \ \exists\, y \in Y : x \prec^* y.$$

The extended alignment graph (3)

Consider the two examples again and define sets $X_1, X_2, \ldots$ via the two given traces:



(a) (b)

In (a), $X_1 \lhd^* X_2 \lhd^* X_3 \lhd^* X_4$, and $\lhd^*$ is a strict partial order.

In (b), we have $X_1 \lhd^* X_2, X_3, X_4$; $X_2 \lhd^* X_3, X_4$; and $X_3 \lhd^* X_2, X_4$. Since, e.g., $X_2 \lhd^* X_2$ does *not* hold, there is no transitivity, and thus the binary relation no partial order.

Characterization of traces

Theorem. (John Kececioglu)
Given a set of sequences A. Let $G = (V, E, H)$ be an extended alignment graph for A. A subset $T \subseteq E$ of edges is a trace, if and only if $\lhd^*$ is a strict partial order on the connected components of $G' = (V, T)$.

(Recall that a *connected component* of a graph is a maximal set of nodes $U \subseteq V$ such that any two nodes $v, u \in U$ are connected by a path of edges in the graph.)

Proof. Blackboard.