# MetaPathway Hunter (MPH)

## Alignment of Metabolic Pathways

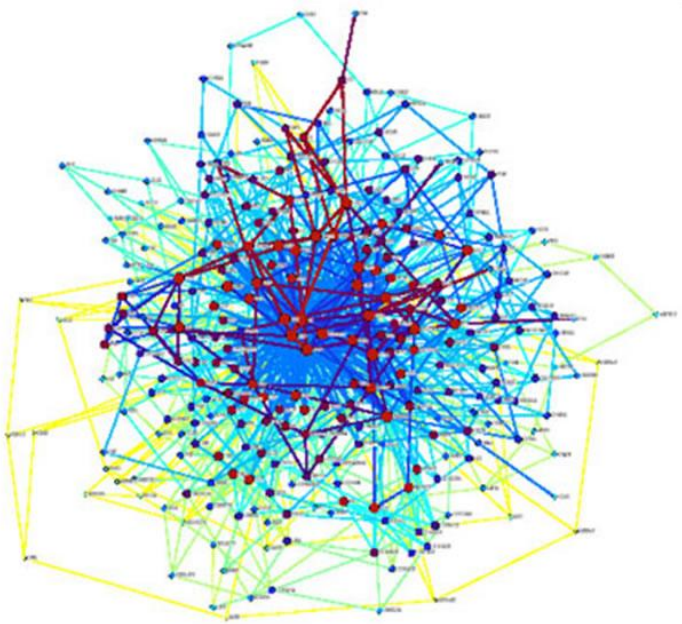
Ron Y. Pinter
Oleg Rokhlenko
Esti Yeger-Lotem
Michal Ziv-Ukelson

# Content

- Metabolic Pathways
- Target
- Representation
- Evaluation Problem
- Approximate Labeled Subtree Homeomorphism Problem (ALSH)
- Meta Pathway Hunter (MPH) As A Tool

# Metabolic Pathways and Network

- Current efforts to reconstruct genome-scale metabolic networks
  - *E. coli*, *S. cerevisiae*, human
- Metabolic pathways databases
  - BioCyc (MetaCyc), KEGG, SGD
  - tools for pathway visualization, queries on pathway components
- Dearth of tools for comparison between pathways

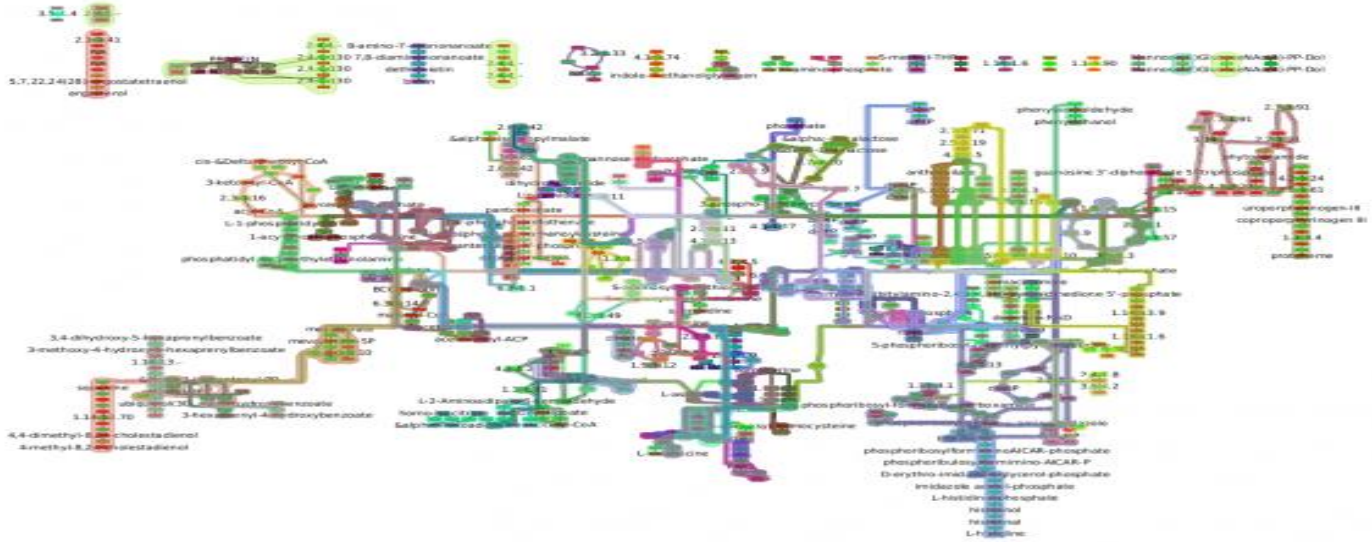


# Targets

- Develop a pathway alignment tool for the detection
  - Novel pathway against a database of pathways.
  - Pathway fragment in a network.
  - Pathway conservation between species.
  - Pathway evolution within a species.
- Pathway similarity (not necessarily identity!) should rely on
  - pathway structure
  - enzyme similarity

# Representation

- A mathematical representation is needed
  - We will use a labeled graph.
  - Nodes are **enzymes**.
  - Labels are the Enzyme Commission (**EC**).
  - Edges are **corresponding enzymes**.



# Evaluation Problem

- Search for repeating patterns in a labeled graph.
- Labeled graphs pattern matching
  - Yet labeled subgraphs isomorphism is a NP-Complete problem (Garey and Johnson, 1979).
- Solution – ALSH approximate labeled subtree homeomorphism.

# Convert Graphs to Trees

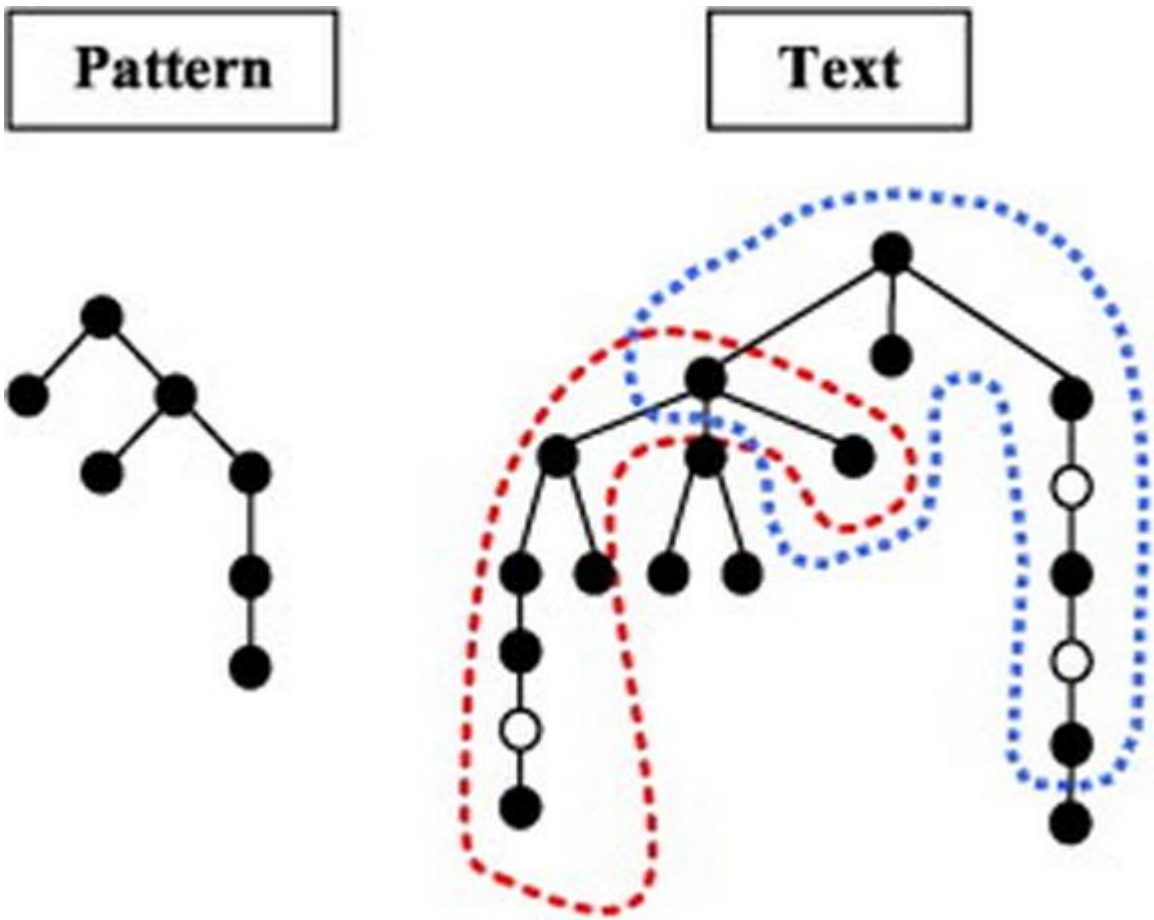
- Most metabolic pathways can be easily cast as a multi source tree or transformed to them without much loss of generality.
- Breaking directed cycles
  - generate alternative multi source trees that cover all possible cycle splitting variations.
  - surprisingly rare (less than $\frac{10}{organizm}$).

- Handling DAGs (Directed acyclic graphs)
  - duplicate and split merge nodes.
  - fits well with biology: *alternative pathways.*

# ALSH Content

- Subtree homeomorphism
- Definitions
- Approach
- Weighted matching (min-cost max-flows)
- Dynamic programing algorithms unordered trees
- ALSH on ordered trees

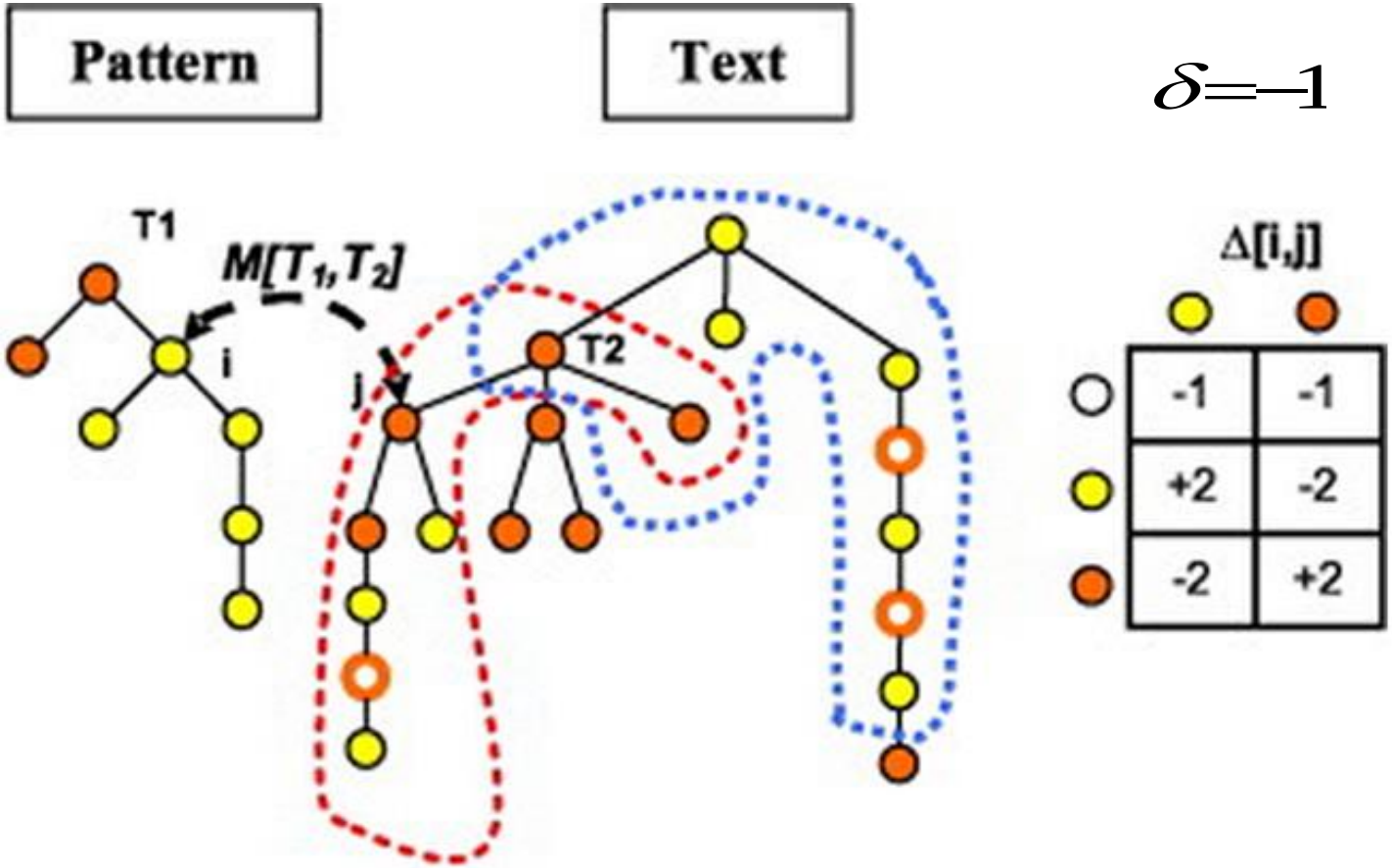
# Subtree Homeomorphism



# Definition I - LSH similarity score

- Let $T_1, T_2$ be 2 labeled trees and $T_1$ is homeomorphic to $T_2$.
- Let $\delta$ be a predefined node deletion penalty.
- Let *M* be a homeomorphism from $T_1$ to $T_2$.
- Let $\Delta$ be scoring table containing a similarity score between any labeled node in $T_1$ and any labeled node in $T_2$.
- Let the LSH similarity score of *M* be

# Best Subtree Homeomorphism




$$LSH(\mathrm{M}) = \delta \cdot \left( |T_2| - |T_1| \right) + \sum_u \Delta\left[u, \mathrm{M}(u)\right]$$

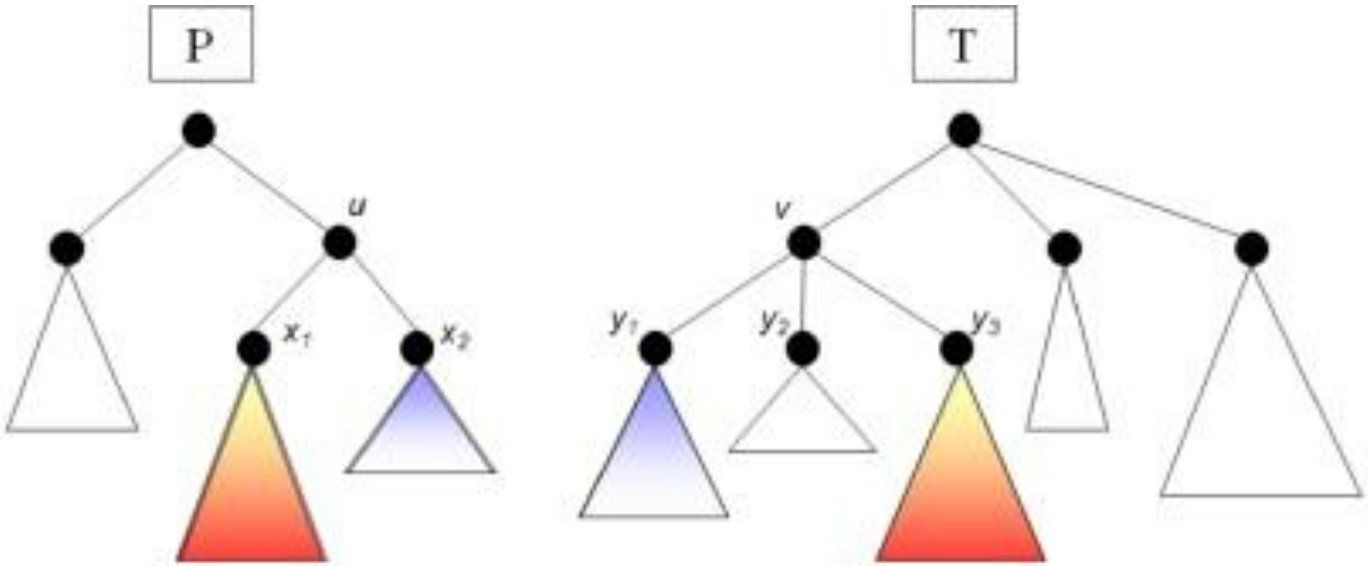
# Definitions II - ALSH

- Let *P,T* be 2 undirected labeled trees.
- ALSH is finding the homeomorphism-preserving mapping *M* from *P* to some subtree *t* of *T* such that LSH(*M*) is maximal.

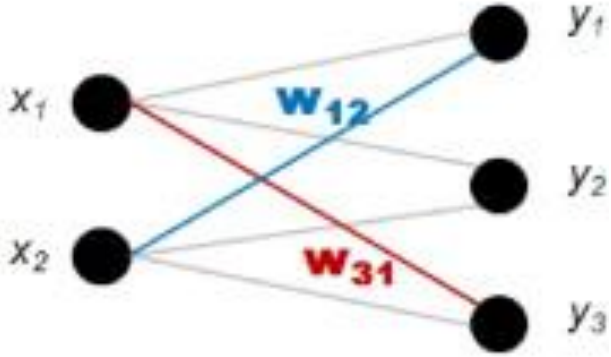
# Solution Approach

- Using a recursive structure to separate the ALSH into smaller ALSH problems.
  - We will use postorder.
  - Starting from the leaves according to $\Delta$ .
- Each pair of internal nodes (u,v) induces a max weight matching problem(Min cost – Max flow).
  - Using bipartite graph $G(X,Y,E)$
  - X is u's children and Y is v's Children.
- The results are combined using dynamic programing.

# Weight Matching




| | $x_1$ | $x_2$ | ... | $u$ | ... |
|---|---|---|---|---|---|
| $y_1$ | $w_{11}$ | $w_{12}$ | | $w_{u1}$ | |
| $y_2$ | $w_{21}$ | $w_{22}$ | | $w_{u2}$ | |
| $y_3$ | $w_{31}$ | $w_{32}$ | | $w_{u3}$ | |
| ... | | | | | |
| $v$ | | | | | |
| ... | | | | | |



# Max Weight Matching Algorithm

- Fredman & Tarjan $\boxed{O(|V||E| + |V|^2 \log(|V|))}$ (using Fibonacci heaps).
- Under the assumption that the input costs are integers in the range [-*C*,…,*0*,…,*C*], Gabow & Tarjan (in "Faster scaling algorithms for network problems") use cost scaling and blocking flow techniques to obtain an $\boxed{O(\sqrt{|V|}|E| \log(|V| \cdot C))}$ time algorithm. **This algorithm is conditioned by the similarity assumption** (i.e. $C = n^k$ for some constant k).
- <u>**Lemma I**</u> - *A flow f is minimum cost* iff *its residual graph R has no negative cost cycle.*

# Dynamic Programing

## Basic Algorithm for Rooted Unordered Trees - RScore

- Let $T(V_T, E_T, r)$ and $P(V_P, E_P, r')$ be 2 text trees which are rooted in r & r'.
- Let $t_v^r$ be a subtree of $T^r$ which is rooted in *v* and contains all of *v*'s descendants. Similarly we will use $p_u^{r'}$ a subtree of $P^{r'}$.
- $\forall v \in V_T, \forall u \in V_P$ the maximal LSH similarity score between $p_u^{r'}$ and some homeomorphic subtree $t_v^r$ will be defined as $RScore[v,u]$ if such subtree does not exist $RScore[v,u] = 0$ .
- $RScore[v,u]$ is calculated recursively in postorder over $T^r$.
- $\forall v \in V_T, \forall u \in V_P$ all of *u*'s children will be $x_1,\ldots,x_{c(u)}$ and all of *v*'s children will be $y_1,\ldots,y_{c(v)}$ .
- let $RScore[v,u] = \max(0, \max_{\text{matching } M}(\sum_{(x,y)\in M} RScore[x,y] + \sum_{(x,y)\notin M} RScore[x,y]))$

**Require:** Rooted trees $T = (V_T, E_T, r)$ and $P = (V_P, E_P, r')$.
**Ensure:** The root of the subtree $t$ of $T$ that has the highest similarity score to $P$, if $T$ has a subtree which is homeomorphic to $P$.

```
for each node u of P in postorder do
    for each node v of T in postorder do
        if u is leaf then
            if v is leaf then
                RScores(v, u) ← Δ[v, u]
            else
                RScores(v, u) ← ComputeScoresForTextNode(v, u)
            end if
        else
            if Level(u) > Level(v) then
                RScores(v, u) ← −∞
            else
                RScores(v, u) ← ComputeScoresForTextNode(v, u)
            end if
        end if
    end for
end for
```

$k > l$

**Procedure** ComputeScoresForTextNode$(v, u)$

```
Let k denote the out-degree of node u and ℓ denote the out-degree of node v.
if k > ℓ then
    AssignmentScore ← −∞
else
    Construct a bipartite graph G with node bipartition X and Y such that X = {x_1, ..., x_k} is the set of children of u, Y = {y_1, ..., y_ℓ} is the
    set of children of v, and every node u_i ∈ X is connected to every node v_j ∈ Y via an edge whose weight is RScores(v_j, u_i).
    Set AssignmentScore to the maximum weight of a matching in G.
end if
BestChild ← max_{j=1}^{ℓ} RScores(y_j, u)
return max{Δ[v, u] + AssignmentScore, BestChild + δ}
```

# Dynamic Programing

**Basic algorithm for rooted unordered trees – best score**

- Let $best\_score$ be the optimal LSH similarity score, it's defined $best\_score = \max_{j=1,\dots,|V_T|} [score[j]]$.
- $\forall v_j \in V_T$ where $score[j] = best\_score$ $v_j$ will be reported as a possible root of a subtree of $T^r$ which has the maximal similarity to P under the LSH similarity score measure.

# Dynamic Programing

**Basic algorithm for rooted unordered trees - complexity**

- Observation I $\sum_{u=1}^{m} c(u) = m-1, \sum_{v=1}^{n} c(v) = n-1$
- For each pair $u \in P, v \in T$ our algorithm calls a function (ComputeScoresForTextNode)
    - The greatest computational complexity comes from calculating a min cost max flow on a bipartite graph.
    - The graphs has $c(v)+c(u)$ nodes.
    - The graphs has $c(v) \cdot c(u)$ edges.
- Using Fredman and Tarjan ’s algorithm in the bipartite graph we compute a match for each one of *u*’s children invoking Dijkstra’s shorted path algorithm.
- $O\left(\sum_{u=1}^{m}\sum_{v=1}^{n}\left(c(v)\cdot c(u)^2 + c(v)\cdot c(u)\cdot \log(c(v))\right)\right) =$

  $O\left(\sum_{u=1}^{m}\left(n\cdot c(u)^2 + n\cdot c(u)\cdot \log(n)\right)\right) = O\left(mn^2 + mn\log(n)\right)$

# Dynamic Programing

## Expansion for unrooted unordered trees - intuition

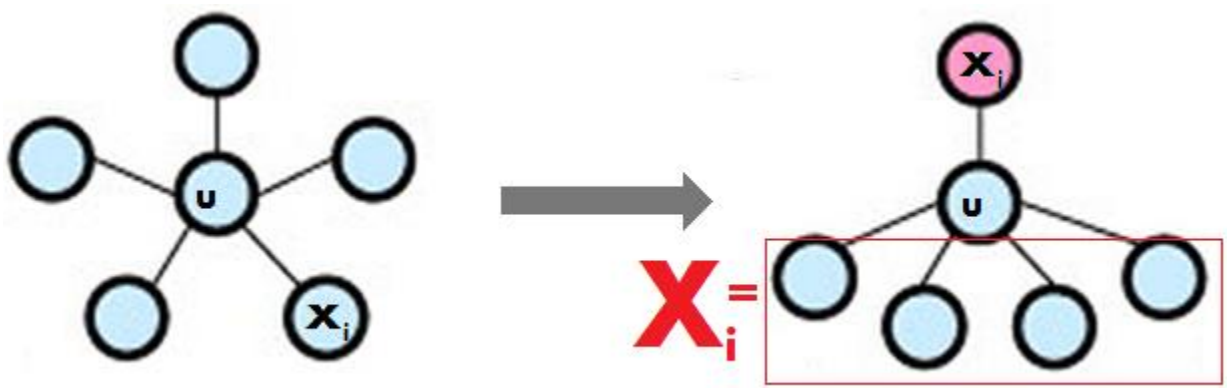
- Why unrooted is more complicated?
  - It seems like $|T| \cdot |P|$ times the previous (rooted) solution.
- How to solve this?
  - We can choose an arbitrary root for one of the trees preferably the bigger one T.
  - We will cover all the options of roots in the pattern tree.

How to calculate it with the same complexity???

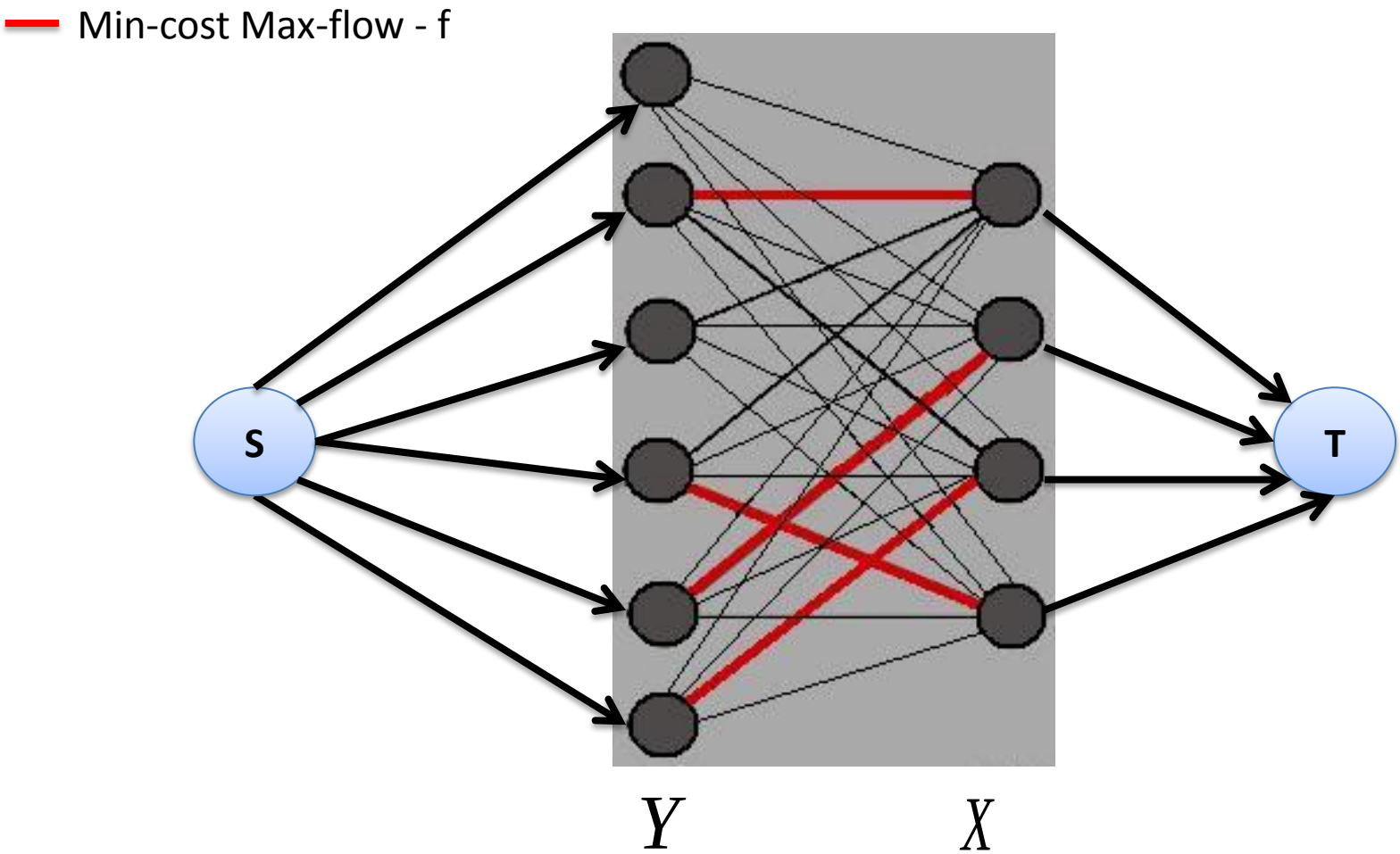
# Dynamic Programing

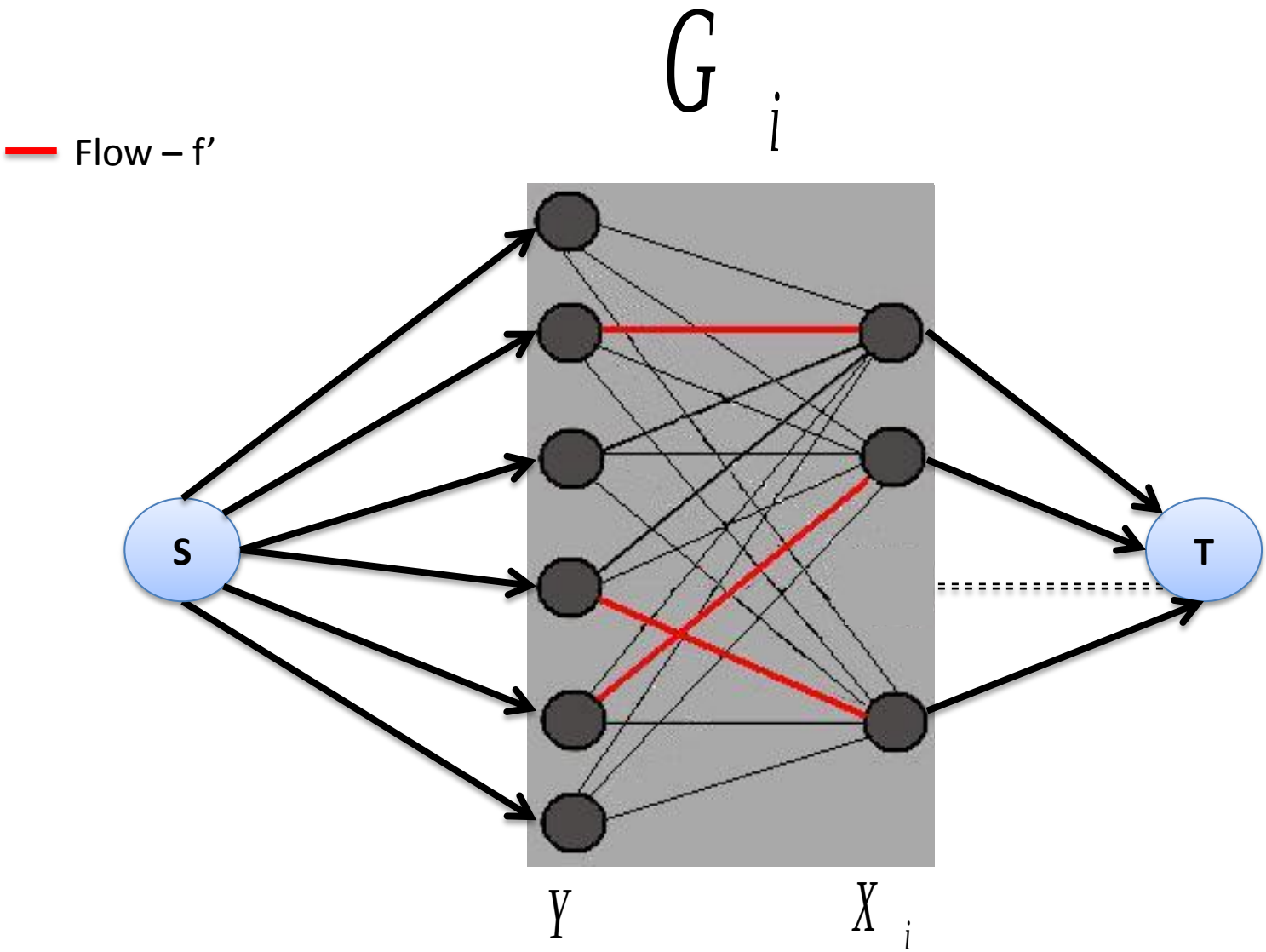
## Expansion for unrooted unordered trees - $G_i$'s

- In unrooted trees there are only neighbors so let $d(u)$ be the number of neighbors of a node $u \in P$ and let $\{x_1, \ldots, x_{d(u)}\} \Rightarrow$ be the neighbors.
- Let $X_i$ be $X_i = \{x \in X \mid x \neq x_i\}$.
- Let $G_i$ be $G_i = (X_i \cup Y, E_i)$.



# $G'$

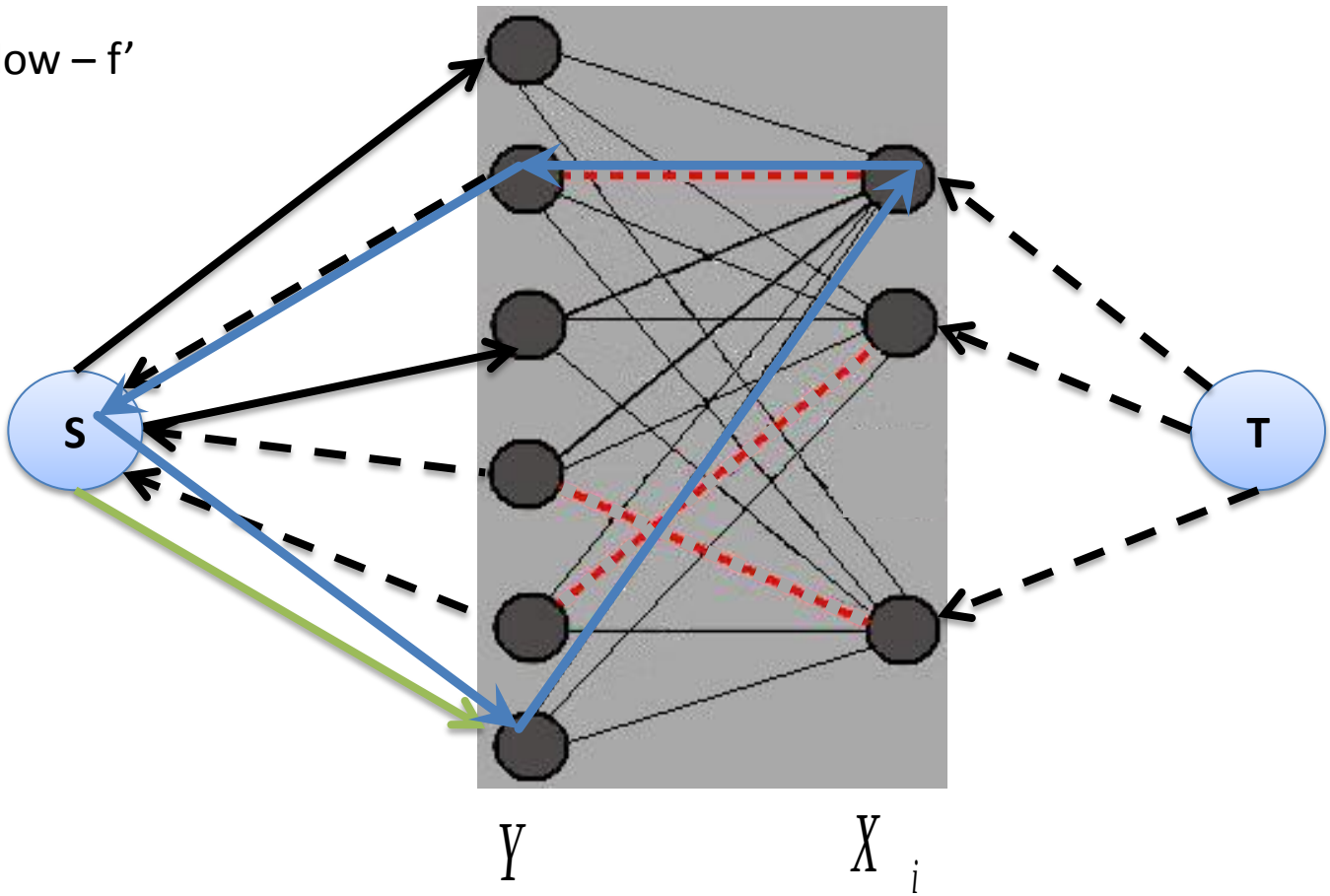



$G_i$
Flow – f'
S
T
$Y$
$X_i$

# $R'$

*Correction path p* - minimal cost cycle which contains .

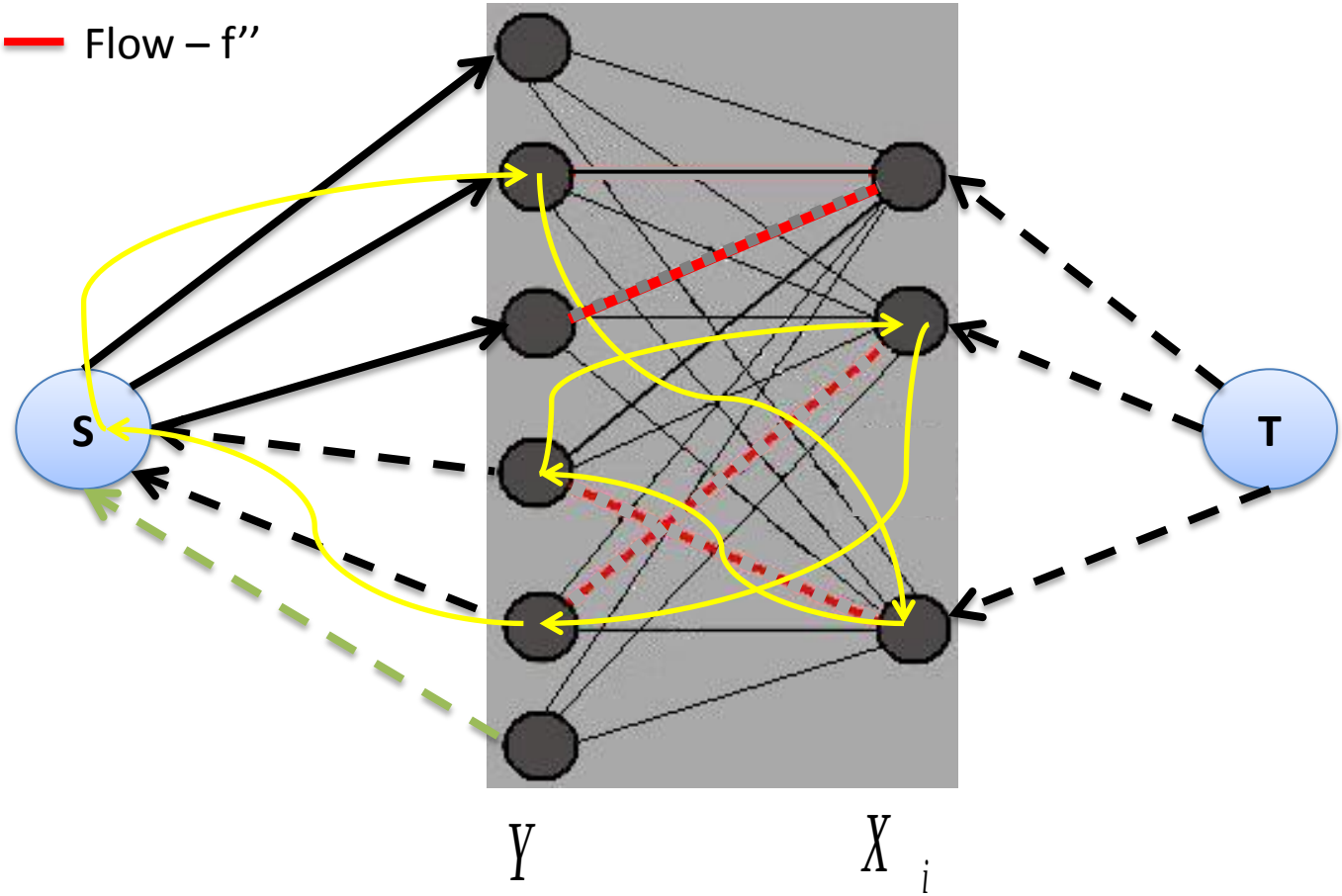
Flow – f'



# $R''$

Claim - the corrected residual graph (flow) has no negative cost cycles (lemma I)

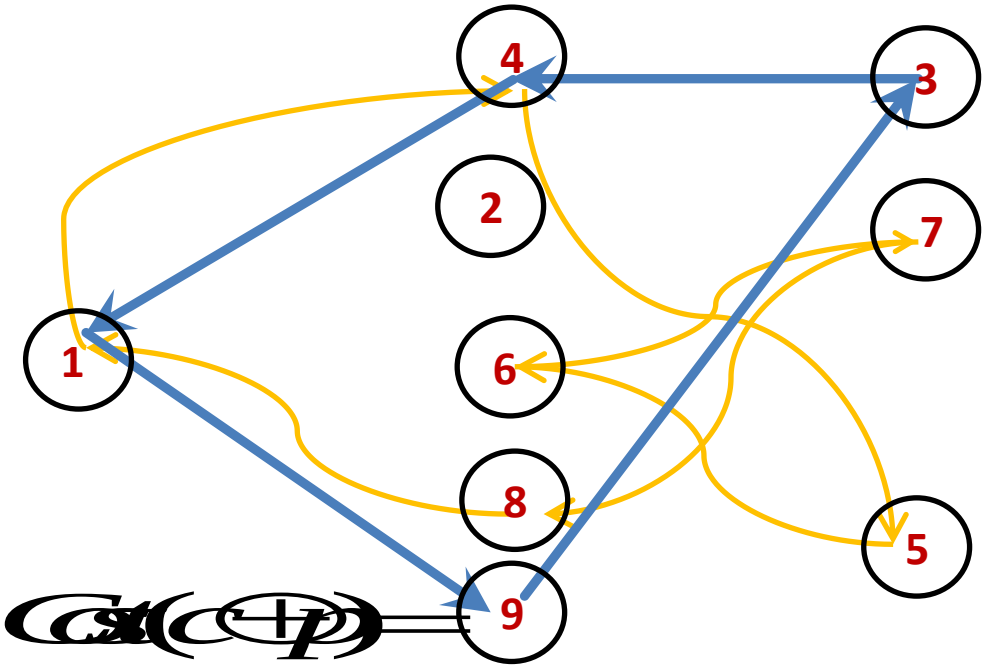
Negative cost cycle *c* - Conversely assume that c exists.



Claim - the corrected residual graph has no negative cost cycles, by **lemma I**: min-cost max-flow

*Correction path p* - minimal cost cycle which contains .

Negative cost cycle *c* - Conversely assumed.




$$cost(c+p) = cost(c) + cost(4,1) + cost(p) + cost(4,1) = cost(c) + cost(p) < cost(p)$$

Contradicts *p*'s minimalism!

# Dynamic Programing

## Expansion for unrooted unordered trees - UScore

- Let $T(V_T, E_T)$, $P(V_P, E_P)$ be 2 labeled trees which are unrooted.
- Let r be an arbitrary root of T.
- $\forall v \in V_P, \forall u \in V_T$ the maximal LSH similarity score between $p_u^{x_i}$ and some homeomorphic subtree $t_v^r$ will be defined as $UScore[v,u,x_i]$.
- If such subtree does not exist $UScore[v,u,x_i] = \epsilon$.
- $UScore[v,u,q]$ is calculated recursively in postorder over $T^r$.
- $\forall v \in V_T, \forall u \in V_P$ all of *u*'s neighbors will be $x_1,\ldots,x_{d(u)}$ and all of *v*'s children will be $y_1,\ldots,y_{c(v)}$.
- Let $Lab[y_i,x_i] = \max(\{Lab(y_i) \text{ matching } x_i$, $\max(\{Lab[y_i,x_i], Lab[y_i,x_i]\})$
- Let $Lab[y_i,x_i] = \text{[illegible]}$

**Require:** Unrooted trees $T = (V_T, E_T)$ and $P = (V_P, E_P)$).
**Ensure:** The root of the subtree $t$ of $T$ which has the highest similarity score to $P$, if $T$ has a subtree which is homeomorphic to $P$.

Pick a vertex $r$ of $T$ to be the root of $T$.
**for** all $u \in P$, $v \in T$ $x_i \in P$ **do**
  UScores$[v, u, x_i] \leftarrow -\infty$.
**end for**
**for** each leaf $v$ of $T_r$ **do**
  **for** each leaf $u$ of $P$ **do**
    UScores$[v, u, \text{parent}(u)] \leftarrow \Delta[v, u]$.
  **end for**
**end for**
**for** each internal node $v$ of $T$ in postorder **do**
  ComputeScoresForTextNode$(v)$
**end for**
*best_score* $\leftarrow \max_{i=1,\ldots,m,\, j=1,\ldots,n}$ UScores$[v_j, u_i, \phi]$

**Procedure** ComputeScoresForTextNode$(v)$
**for** each node $u$ of $P$ **do**
  BestChild$(v, u, x_i) \leftarrow \max_{j=1,\ldots,\ell}$ UScores$[y_j, u, x_i]$
  Construct a bipartite graph $G$ with node bipartition $X$ and $Y$ such that $X = \{x_1, \ldots, x_k\}$ is the set of neighbors of $u$, $Y = \{y_1, \ldots, y_\ell\}$ is the set of children of $v$, and every node $x_i \in X$ is connected to every node $y_j \in Y$ via an edge whose weight is UScores$[y_j, x_i, u]$.
  Let $X_0 = X$ and $X_i = X - \{x_i\}$.
  **for** all $1 \leqslant i \leqslant k$ **do**
    **if** Level$(v) <$ Level$(u, x_i)$ **then**
      UScores$[v, u, x_i] \leftarrow -\infty$
    **else**
      **if** $k > \ell + 1$ **then**
        AssignmentScore$(X_i, Y) \leftarrow -\infty$.
      **else**
        Compute the score AssignmentScore$(X_i, Y)$ of the maximum weight matching in $G_i$.
      **end if**
      UScores$[v, u, x_i] \leftarrow \max\{\Delta[v, u] + \text{AssignmentScore}(X_i, Y), \text{BestChild}(v, u, x_i) + \delta\}$
    **end if**
  **end for**
  **if** $k > \ell$ **then**
    UScores$[v, u, \phi] \leftarrow -\infty$
  **else**
    UScores$[v, u, \phi] \leftarrow \Delta[v, u] +$ maximum weight matching in $G$.
  **end if**
**end for**

*leafs only*

*Indirected*

$k > l + 1$

$k > l$

# Dynamic Programing

**Expansion for unrooted unordered trees – best score**

- Let $best\_score$ be the optimal LSH similarity score, it's defined [illegible].
- $\forall v \in V_T, \forall u \in V_P$ where [illegible], $v$ will be reported as a possible root of a subtree of $T^r$ which has the maximal similarity to $P^u$ under the LSH measure.

# Dynamic Programing

## Expansion for unrooted unordered trees – Complexity I

- Bottleneck: $G_1, G_2, G_{d(u)}$ seems $d(u)$ times more complicated than solving one $G$.
- The solution is that from G's min-cost max-flow we can compute in $O(|E|+|V|\log|V|)$ all of the $G_i s'$ flow solutions.
- Observation II $\sum_{u=1}^{m} d(u) = 2m-2$.
- For each pair $u \in P, v \in T$ our algorithm calls a function (ComputeScoresForTextNode)
    - The greatest computational complexity comes from calculating a min cost max flow on $d(u)$ bipartite graphs.
    - The graphs has $d(v)+d(u)$ nodes.
    - The graphs has $d(v)\cdot d(u)$ edges.

# Dynamic Programing

## Expansion for unrooted unordered trees – Complexity II

- Our bottleneck can be solved in $O\left(d(u)\cdot(E+V\log(V))\right)$
  - Using Fredman and Tarjan ’s algorithm in the bipartite graph $G$ we compute a match for each one of *u*’s neighbors invoking Dijkstra’s shorted path algorithm $O\left(d(u)\cdot(E+V\log(V))\right)$ run time complexity.
  - For each $x_i \in X$ we create a bipartite graph $G_i$ a match can be calculated from $G's$ matching using a one run of a single source shortest path algorithm $O\left(d(u)\cdot(E+V\log(V))\right)$ run time complexity.
- Therefore the total run time upper bound is

$$O\left(\sum_{u=1}^{m}\sum_{v=1}^{n}\left(d(v)\cdot d(u)^2+d(v)\cdot d(u)\cdot\log(d(v))\right)\right)\overset{obs1}{=}$$

$$O\left(\sum_{u=1}^{m}\left(n\cdot d(u)^2+n\cdot d(u)\cdot\log(n)\right)\right)\overset{obs2}{=}O\left(m\cdot n^2+m\cdot n\cdot\log(n)\right)$$

# Dynamic Programing

## Expansion for unrooted unordered trees – Complexity II

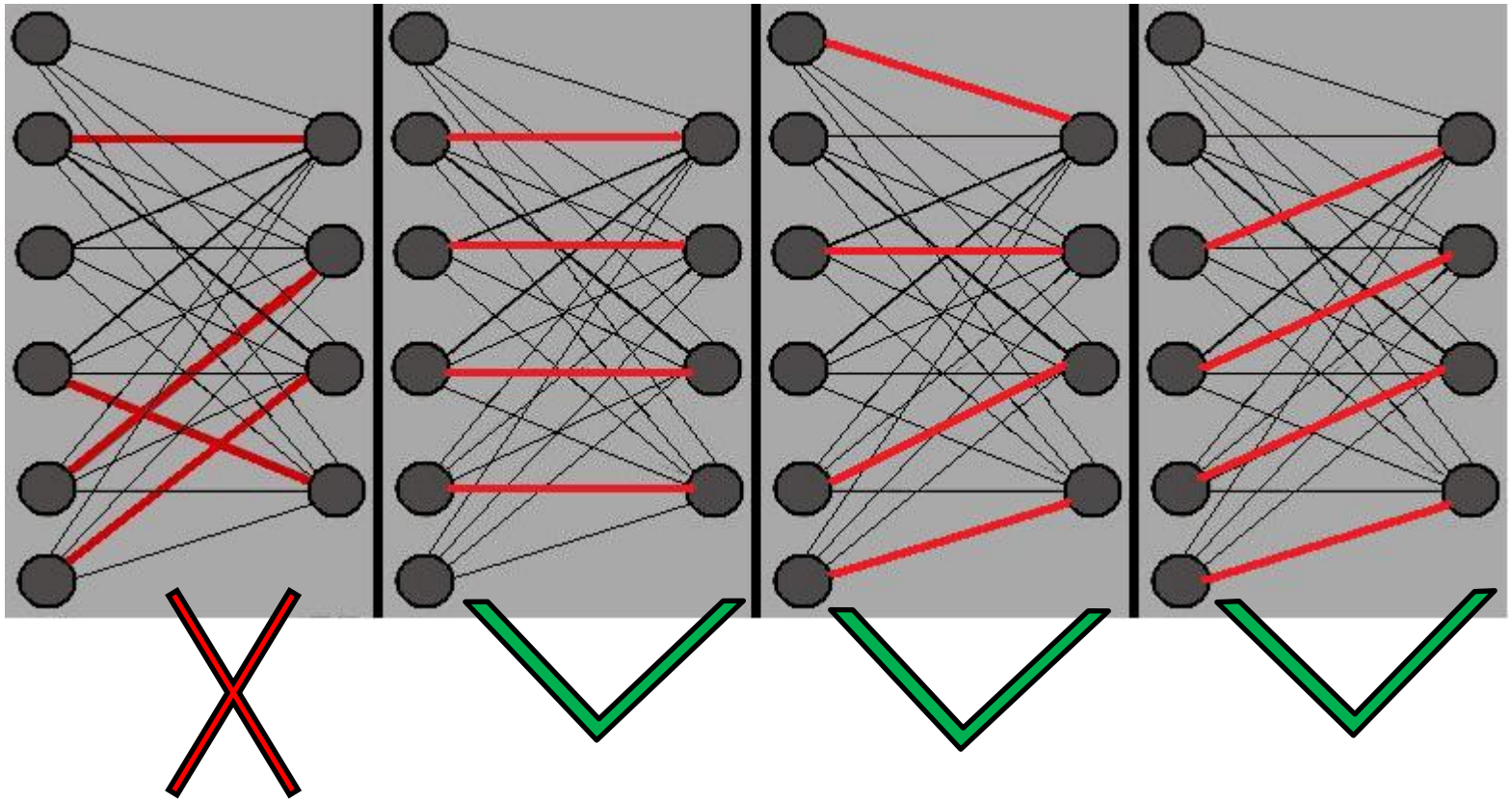
- Our bottleneck can be solved in $O\left(d(u)\cdot(E+V\log(V))\right)$
  - Using Fredman and Tarjan ’s algorithm in the bipartite graph $G$ we compute a match for each one of *u*’s neighbors invoking Dijkstra’s shorted path algorithm $O\left(d(u)\cdot(E+V\log(V))\right)$ run time complexity.
  - For each $x_i \in X$ we create a bipartite graph $G_i$ a match can be calculated from $G's$ matching using a one run of a single source shortest path algorithm $O\left(d(u)\cdot(E+V\log(V))\right)$ run time complexity.
- Therefore the total run time upper bound is

$$O\left(\sum_{u=1}^{m}\sum_{v=1}^{n}\left(d(v)\cdot d(u)^2+d(v)\cdot d(u)\cdot\log(d(v))\right)\right)\overset{obs1}{=}$$

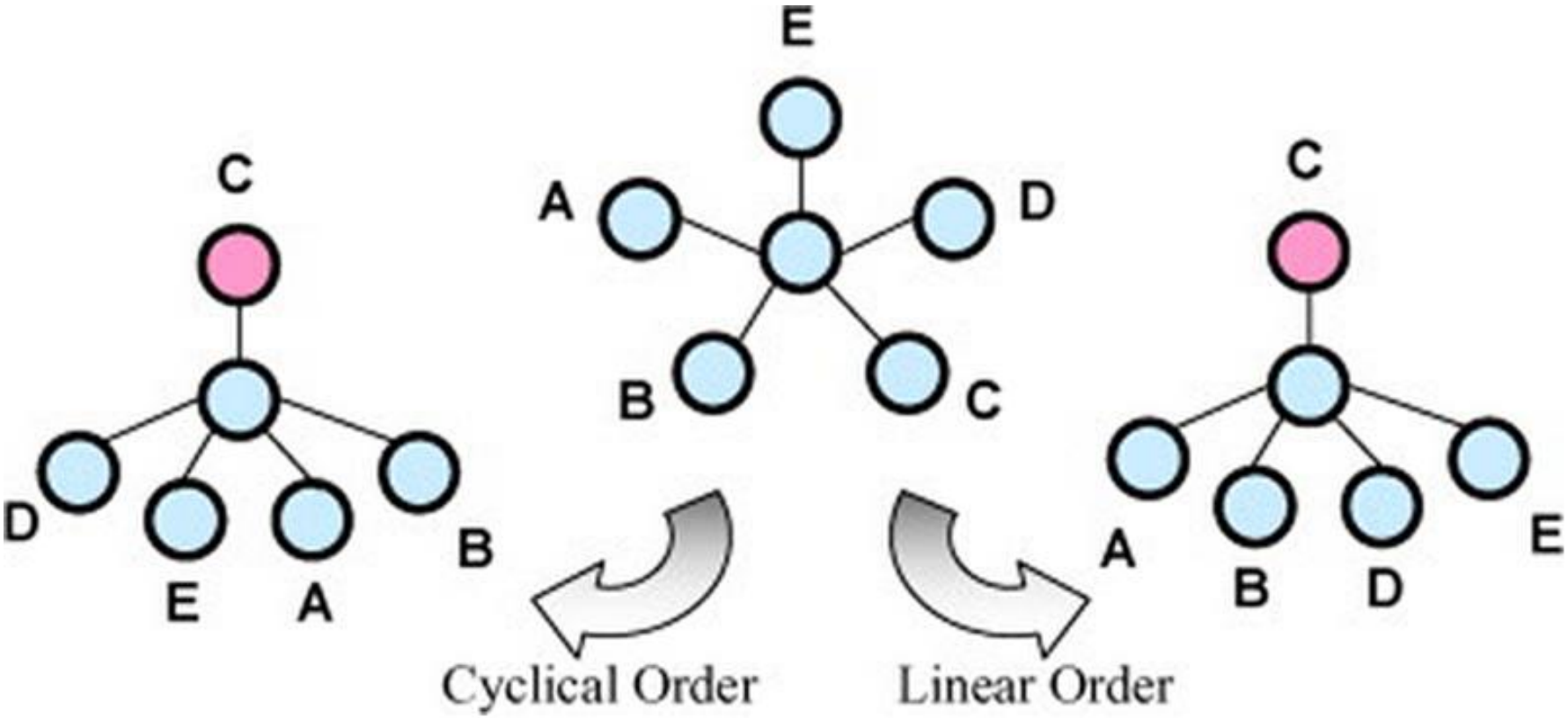
$$O\left(\sum_{u=1}^{m}\left(n\cdot d(u)^2+n\cdot d(u)\cdot\log(n)\right)\right)\overset{obs2}{=}O\left(m\cdot n^2+m\cdot n\cdot\log(n)\right)$$

# Ordered rooted trees

- It exactly like comparing 2 strings in the matching.
- it's complexity is $O(c(v)+c(u))$



# Ordered unrooted trees



# ALSH Summary

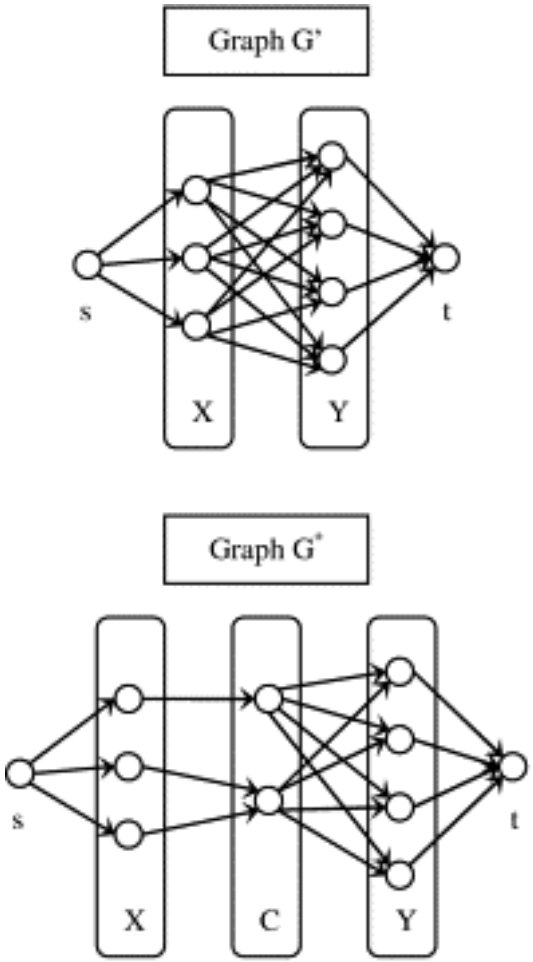
| ALSH running time | ordered | unordered |
|---|---|---|
| rooted | $O(n \cdot m)$ | $O(n \cdot m^2 \cdot n \cdot \log n)$ |
| unrooted | cyclic: $O(n \cdot m \cdot \log(n))$ / linear: $O(n \cdot m)$ | $O\left(n \cdot m^2 + n \cdot m \cdot \log n\right)$ |

**m - # of vertices in P**

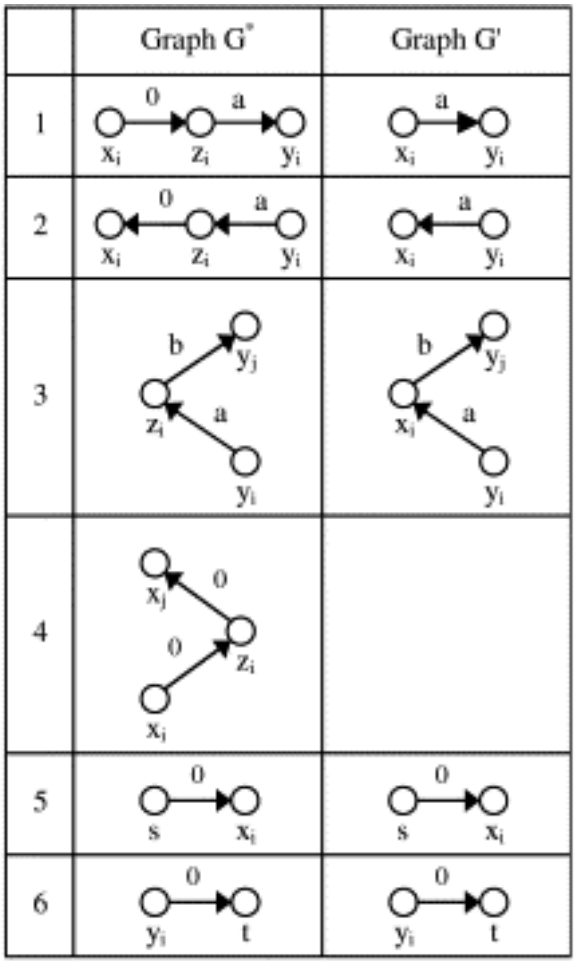
**n - # of vertices in T**

# Dynamic Programing

**more efficient ALSH algorithm for unordered trees**




A




B

# Dynamic Programing

**more efficient ALSH algorithm for unordered trees**

- Lemma:
  - The matching between $u \in V_p$ and $v \in V_T$ can be computed in time: $E + V \lg V$
- where
  - $d(u)$ is the number of neighbors of *u.*
  - $D(u)$ is the number of distinct trees in the forest of trees rooted at neighbors of *u.*
  - $c(v)$ is the number of children of *v.*

# Dynamic Programing

**more efficient ALSH algorithm for unordered trees**

- For each pair $u \in P, v \in T$ our new algorithm will work in:

$$O\left(d(u)\cdot\left(clusters_u \cdot c(v) + c(v)\cdot\log(c(v))\right)\right)$$

- The total sum for all the pairs is :

$$O\left(\sum_{u=1}^{m}\sum_{v=1}^{n}\left(d(u)\cdot\left(clusters_u \cdot c(v) + c(v)\cdot\log(c(v))\right)\right)\right) \overset{obs.1}{=}$$

$$O\left(\sum_{u=1}^{m}\left(d(u)\cdot clusters_u \cdot n + d(u)\cdot n\cdot\log(n)\right)\right) \overset{obs.2}{=}$$

$$O\left(n\cdot\sum_{u=1}^{m}\left(d(u)\cdot D(u)\right) + m\cdot n\cdot\log(n)\right)$$

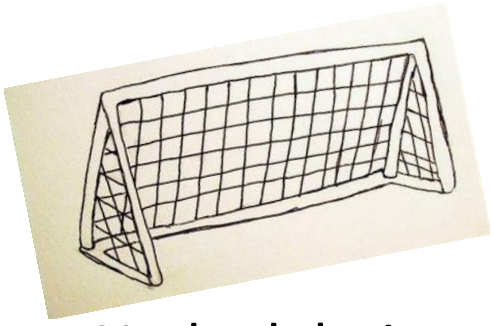
# Dynamic Programing

**more efficient ALSH algorithm for unordered trees**

- Lemma: $\sum_{u=1}^{m} d(u) \cdot L(u) = O\left(\frac{m^2}{\log m}\right)$

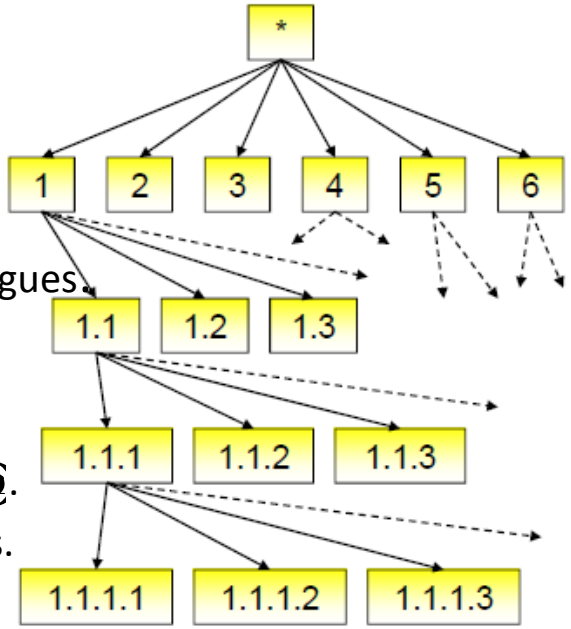
  (similar to [Feder and Motwani 1991, Shamir and Tsur 1999] )

- Therefore the optimal *ALSH* solution for two rooted unordered trees run-time complexity is $O\left(\frac{n^2}{\log n} + n \log n\right)$



# Alignment scoring

- Node deletion score, modeling gaps in the pattern, which entails a fixed penalty $\delta$ , **a parameter for MPH**.
  - A low value encourages alignments of evolutionary remote pathways, where only bits of the pathway are conserved.
  - A higher value encourages matching further enzymes to avoid gaps.
- Node substitution scores that are rated by a label substitution table.
  - **EC** (Enzyme Commission) :
    - Numbering system of enzymes.
    - Each enzyme is represented by a set of 4 numbers.
    - A functional classification of enzymes.
    - Enzymes with similar classification are functions homologues.
  - For an enzyme class *h*.
    - $C(h)$ denotes the number of enzymes included under *h*.
    - $I(h)$ the information content of *h is* $I(h) = \log_2(C(h))$.
    - For enzymes $e_i, e_j$ , $h_{ij}$ is the lowest common upper class.
    - The similarity score of $e_i, e_j$ is $I(h_{ij})$.



# Statistical Significance

- The p-value cutoff used in the analysis is 0.01.
- Pathway pairs with at least one statistically significant alignment between them as significantly aligned.
- To assess whether the number of significantly aligned pathway pairs in the comparisons deviate significantly from the number expected by pure chance at a cutoff of 0.01.
- The binomial test $(k, n, p)$ is used per comparison to ensure. This test computes the probability of having at least k (successes) significantly aligned pairs in n (experiments) total number of aligned pathway pairs with probability p(=0.01) for success.
- This test was performed using the R project for Statistical Computation.

# MetaPathway Hunter As A Tool

- Searches a pattern **P** against a database of labeled trees $F$ .
- Each alignment is given a score ***s*** and statistical significance ***p***.
- Statistical significance ***p*** computation:
  - Executes **P** against a database of 100 randomized $F$ .
  - $p(P)$: the percentage of randomized $F$ which scored more than ***s.***
  - ***p*** = $p(P) = \frac{\#\text{graphs}}{\#\text{randomgraphs}}$
- Enables visualization of the alignment, which is color-coded by similarity.

Each random graph has the same nodes with the same degrees and the same number of edges

# Implementation Details

- Implementation
  - Code in C++
  - Java-based GUI to allow usage as a website.
- System requirements
  - Any Intel Pentium-based computers.
  - Microsoft Windows operating system (version 2000 and higher).
  - No further requirements (hardware and software).
- Query response
  - The 5 best matches per pathway sorted by score & statistical sig.
  - An HTML file as a visual aid with the query drawn on the aligned MPs.
- Query input
  - The penalty score $\delta$.
  - Pattern ***P*** in the processed tree form.
  - A number of metabolic pathways (MPs) ***T*** (it can accept more than one ***T***) in the processed tree form.

**How will we choose the penalty score?**

# Performance

- All against all alignments were run between E.coli (prokaryotic) and S.cerevisiae (eukaryotic) .
- Took 3.66 hours on a Pentium 4, 2.6 GHz clock, 512 MB RAM computer.
- An average of 47 s per query.

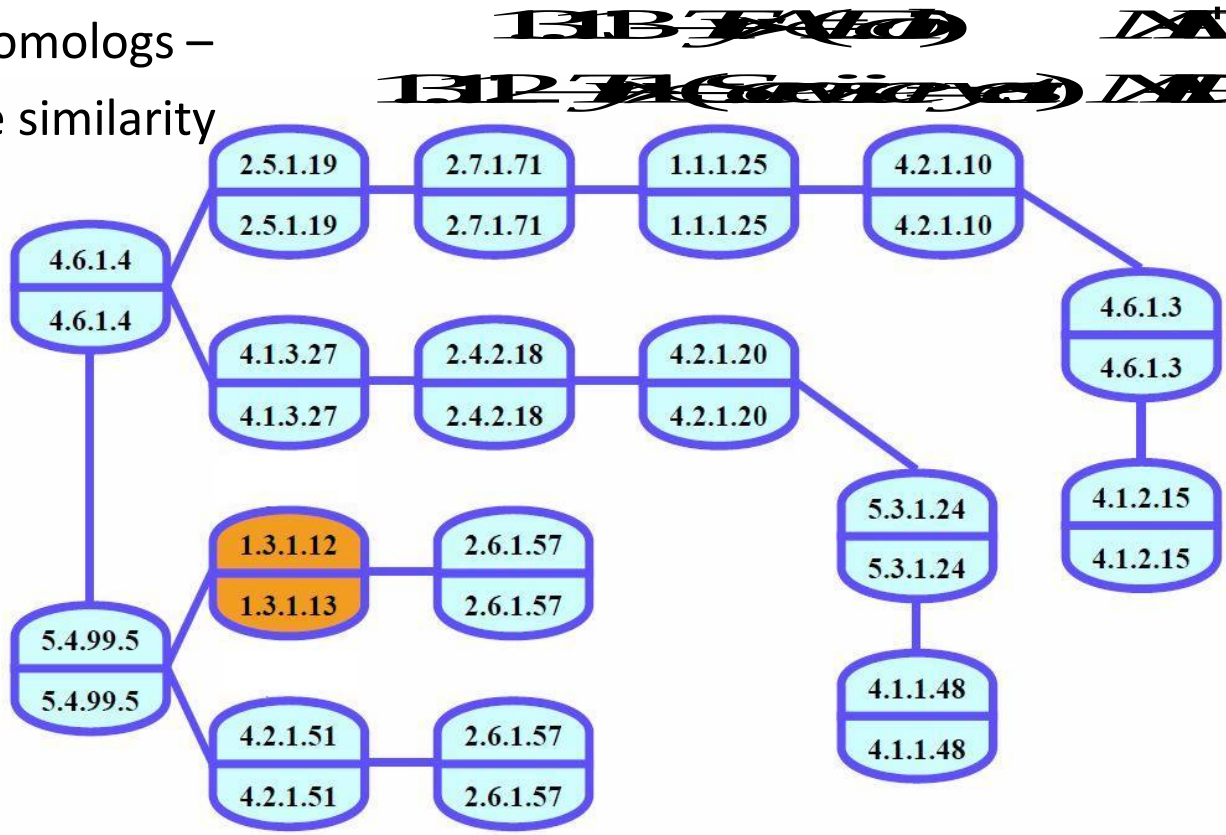
# Inter-Species Alignments I

- Conservation of metabolic pathways:
  - The alignments between 62 out of 80 pairs of analogous pathways were significant.
  - Conservation is not limited to small pathways, *e.g.*

    functional homologs –

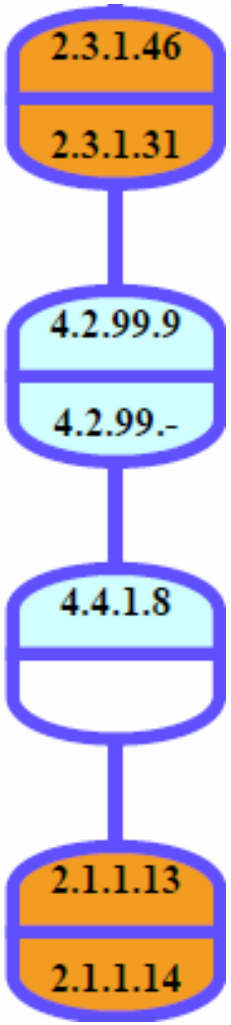
    no sequence similarity

Biosynthesis of phenylalanine, tyrosine, and tryptophan, $s$=-4.28; $p$<0.01



# Inter-Species Alignments II

- Alignment with gaps
  - A gap found
  - Both sequences do the same thing.
    - May be gene fusion in yeast.
    - Or gene duplication in E.coli.

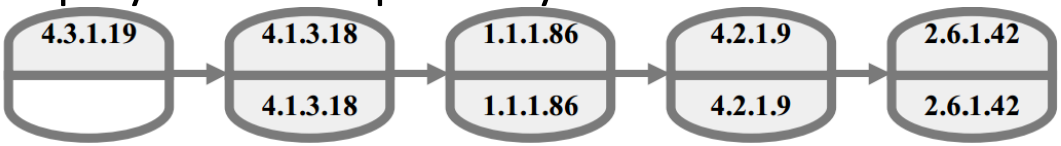




Homoserine and methionine biosynthesis, *E. coli* vs. yeast: $s$=-13.15, $p$<0.01

# Intra-Species Alignments

- Hints to evolution of paralogous pathways
- Amino acid biosynthesis pathways are conserved
- Valine, Leucine, Isoleucine have similar biosynthesis pathways.
  - Valine vs. isoleucine employ the same pathway but on different substrates

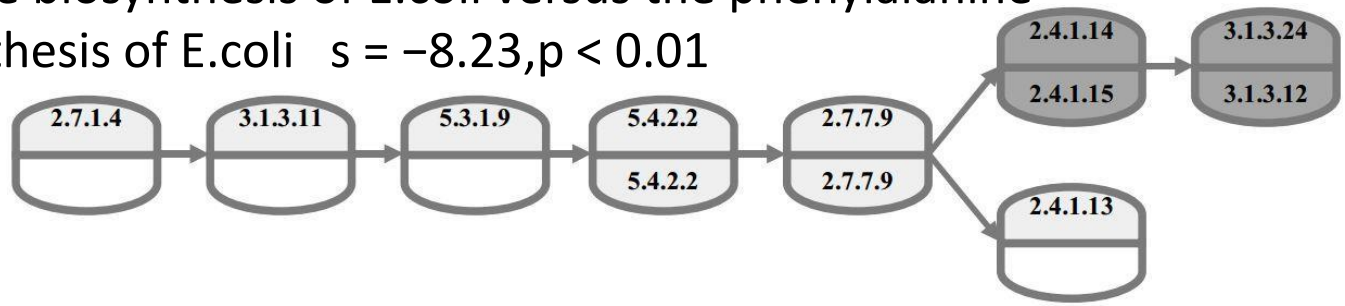
    $s = 0, p < 0.01$




  - The trehalose anabolism pathways of yeast versus the sucrose biosynthesis pathway of yeast

    $s = -9.58, p < 0.01$


  - tyrosine biosynthesis of E.coli versus the phenylalanine biosynthesis of E.coli $s = -8.23, p < 0.01$



# MetaPathway Queries

## Flexible searching – possible queries

- Search for a metabolic connection between enzymes (source and product query).
- Search for a metabolic pathway using only a part of it (common pathway core query)