KO

- PROBLÉMY KOMBINATORICKÉ OPTIMALIZACE ZAHRNUJÍCÍ POPIS APLIKACÍ, FORMALIZACI PROBLÉMŮ, ROZBOR SLOŽITOSTI A ŘEŠÍCÍ ALGORITMY

- INTEGER LINEAR PROGRAMMING
  - ILP
  - PROBLEM GIVEN BY
    - MATRIX $A \in \mathbb{R}^{m \times n}$
    - AND VECTORS $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$
  - GOAL IS TO FIND VECTOR $x \in \mathbb{Z}^n$
  - SUCH THAT
    $A \cdot x \leq b$
    $c^T \cdot x$ IS MAXIMUM
  - ILP DIFFERS FROM LP BY ALLOWING ONLY INTEGER-VALUED SOLUTIONS
    - WE CALL PROBLEM ILP IF AT LEAST ONE VARIABLE IS INTEGER-ONLY
  - BY SOLVING LP AND ROUNDING THE SOLUTION, WE CAN GET SUBOPTIMAL SOLUTION OR NON-FEASIBLE SOLUTION
  - LP IS SOLVABLE IN POLYNOMIAL TIME
  - ILP IS NP-HARD

- 2 - PARTITION PROBLEM

- NUMBER OF BANKNOTES $n \in \mathbb{Z}^+$
- THEIR VALUES $p_1, \dots, p_n \in \mathbb{Z}^+$
- IS THERE SUBSET $S \subseteq \{1, \dots, n\}$ SUCH THAT

$$\sum_{i \in S} p_i = \sum_{i \notin S} p_i$$

- ILP FORMULATION

MIN 0

St $\sum_{i \in 1 \dots n} x_i \cdot p_i = 0{,}5 \cdot \sum_{i \in 1 \dots n} p_i$

$n \in \mathbb{Z}^+$, $p_{i \in 1 \dots n} \in \mathbb{Z}^+$

$x_{i \in 1, \dots, n} \in \{0, 1\}$

- ILP OPTIMIZATION VERSION
  - WE INTRODUCE THRESHOLD
  - IF THERE IS NO SOLUTION, THRESHOLD WILL BE AS CLOSE AS POSSIBLE TO IT

MIN C_{MAX}

St $\sum_{i \in 1 \dots n} x_i \cdot p_i \leq C_{MAX}$

$\sum_{i \in 1 \dots n} (1 - x_i) \cdot p_i \leq C_{MAX}$

$n \in \mathbb{Z}_0^+$, $p_{i \in 1 \dots n} \in \mathbb{Z}_0^+$

$x_{i \in 1 \dots n} \in \{0, 1\}$, $C_{MAX} \in \mathbb{R}_0^+$

- SHORTEST PATHS

  - DIGRAPH G WITH n NODES
  - DISTANCE MATRIX $C: V \times V \to \mathbb{R}_0^+$
  - NODES $s, t \in V$
  - GOAL IS TO FIND SHORTEST PATH FROM s TO t IF THEY ARE REACHABLE

$$\text{MAX } l_t$$

$$\text{st } l_s = 0$$

$$l_j \leq l_i + c_{i,j} \quad i \in 1 \dots n, \; j \in 1 \dots n$$

$$n \in \mathbb{Z}_0^+, \; c_{i \in 1 \dots n, j \in 1 \dots n} \in \mathbb{R}_0^+$$

$$l_{i \in 1 \dots n} \in \mathbb{R}_0^+$$

- TSP

$$\text{MIN} \sum_{i \in 1 \dots n} \sum_{j \in 1 \dots n} c_{i,j} \cdot x_{i,j}$$

$$\text{SE} \sum_{i \in 1 \dots n} x_{i,j} = 1 \quad j \in 1 \dots n \quad \text{- ENTER EACH VERTIX ONCE}$$

$$\sum_{j \in 1 \dots n} x_{i,j} = 1 \quad i \in 1 \dots n \quad \text{- LEAVE EACH VERTEX ONCE}$$

$$s_i + c_{i,j} - (1 - x_{i,j}) \cdot M \leq s_j \quad \text{- CYCLE INDIVISIBILITY}$$

$$M \in \mathbb{Z}_0^+, \; n \in \mathbb{Z}_0^+, \; c_{i \in 1 \dots n, j \in 1 \dots n} \in Q^+$$

$$x_{i \in 1 \dots n, j \in 1 \dots n} \in \{0,1\}, \; s_{i \in 1 \dots n} \in \mathbb{R}_0^+$$

- ENUMERATIVE METHODS
  - INSPECTION OF ALL POSSIBLE SOLUTIONS
  - SUITABLE FOR SMALL INSTANCES

- BRANCH AND BOUND
  - SPLITTING SOLUTION SPACE INTO DISJOINT SETS
  - RELAXES INTEGRALITY AND SOLVES LP FIRST
    - IF ALL VARIABLES ARE INTEGERS, WE END
    - OTHERWISE ONE VARIABLE IS SELECTED AND ITS VALUE IS ASSIGNED TO k
  - WE DIVIDED SPACE INTO TWO SETS
    - $x_i \leq \lfloor k \rfloor$
    - $x_i \geq \lfloor 2 \rfloor + 1$
  - THE ALGORITHM IS RUNNER RECURSIVELY UNTILL WE FIND THE SOLUTION
  - BRANCHING CREATES TREES
    - INNER VERTEX IS SOLUTION OF LP
    - LEAF IS SOLUTION OF ILP
  - AS SOON AS WE FIND ILP SOLUTION WE CAN USE ITS OBJECTIVE VALUE TO CUT PARTS OF TREES WITH WORSE OBJECTIVE VALUE

- UNIMODULAR MATRIX

  - SPECIAL CASE IN WHICH ILP CAN BE SOLVED IN POLYNOMIAL TIME

  - MATRIX $A = [a_{ij}]$ OF SIZE m/n IS TOTALLY UNIMODULAR IF THE DETERMINANT OF EVERY SQUARE SUB MATRIX OF MATRIX A IS EQUAL $0, 1, -1$

    - NECESSARY CONDITION

      - IF A IS UNIMODULAR THEN $a_{ij} \in \{0, 1, -1\}\ \forall i,j$

    - SUFFICIENT CONDITION

      - EACH COLUMN IN A CONTAINS ONE NON-ZERO ELEMENT OR EXACTLY TWO NON-ZERO ELEMENTS +1 AND -1

- CUTTING PLANES METHOD

  - SIMILAR TO BRANCH AND BOUND

    - REPEAT SOLUTION OF LP

  - ITERATIVELY ADDS CONSTRAINTS THAT CUT OFF PART OF SOLUTION SPACE

  - NEW CONSTRAINTS MUST FULFILL CONDITIONS

    - SOLUTION FOUNDED BY LP BECOMES INFEASIBLE

    - ALL INTEGER SOLUTIONS ARE FEASIBLE IN THE CASE STEP HAVE TO REMAIN FEASIBLE

- SHORTEST PATHS
  - GRAPH G
  - WEIGHTS $c: E(G) \to \mathbb{R}$
  - NODES $s, t \in V(G)$
  - FIND SHORTEST PATH P FROM s TO t
    - $c(E(P))$
    - OR DECIDE THAT t IS UNREACHABLE FROM s.

- ANOTHER FORMULATIONS
  - FROM s TO ALL NODES
  - FROM EVERY NODE TO t
  - BETWEEN ALL PAIRS OF NODES

- LONGEST PATH
  - TRANSFORM TO PROBLEM OF SHORTEST PATH BY
    - REVERSE SIGN OF WEIGHTS
    - SEARCH FOR MINIMUM

- WEIGHTED NODES

- MINIMUM SPANNING TREE
  - FIND SET OF EDGES WITH MINIMAL COST, SO THAT GRAPH IS CONNECTED

- NEGATIVE WEIGHTS
  - THEY ARE ALLOWED
  - NEGATIVE CYCLES ARE NOT ALLOWED
    - SUCH PROBLEM BECOMES NP-HARD
  - IF WE TRANSFOR NON-DIRECTED TO DIRECTED GRAPH WE CAN'T USE DIRECTED NEGATIVE WEIGHTS
    - THIS WOULD CREATE NEGATIVE CYCLE

- EXISTENCE OF SHORTEST PATH
  - IF PATH FROM s TO t EXISTS
    - THAN SHORTEST PATH FROM s TO t EXISTS TOO
  - PATH CAN'T REPEAT EDGES
    - OTHERWISE WE COULD REPEAT EDGE WITH NEGATIVE WEIGHT AND RECIEVE BETTER AND BETTER SOLUTION AND NEVER OBTAINING MINIMUM
  - LENGTH OF PATH P IS SUM OF ITS WEIGHTS
  - DISTANCE FROM s TO t $\ell(s,t)$ IS LENGTH OF SHORTEST PATH

- TRIANGLE INEQUALITY

  - IF GRAPH DOESN'T CONTAIN A NEGATIV CYCLE OF NEGATIVE WEIGHT THAN

$$\ell(i,j) \leq \ell(i,k) + \ell(k,j)$$

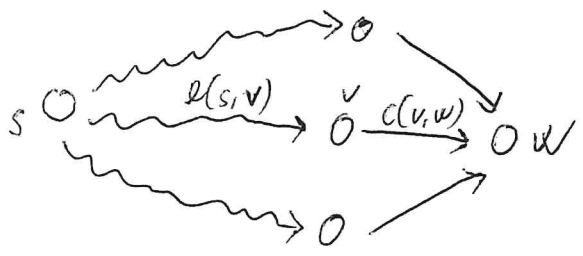
- BELLMAN'S PRINCIPLE OF OPTIMALITY

  - WE HAVE DIGRAPH G WITH NO NEGATIVE CYCLES
  - LET $k \in \mathbb{N}$ AND s, w BE TWO VERTICES
  - LET P^k BE SHORTEST PATH AMONG ALL s-w-PATHS WITH AT MOST k EDGES
  - LET $e = (v, w)$ BE ITS FINAL EDGE
  - THAN $P^{k-1}[s, v]$ (P WITHOUT EDGE e) IS A SHORTEST ONE AMONG ALL s-v-PATHS WITH AT MOST $k-1$ EDGES

- BELMAN'S EQUATION

$$\ell(s, w) = \min_{v \neq w} \{\ell(s, v) + c(v, w)\}$$

- HOLDS IF GRAPH DOESN'T HAVE NEGATIVE CYCLES



- GENERALIZATION
  - IF WE HAVE SHORTEST PATH $S-W$ GOING THROUGH NODE V, WE CAN DIVIDE IT INTO SHORTEST PATH FROM S TO V AND FROM V TO W

$$\ell(s,w) = \ell(s,v) + \ell(v,w)$$

- DIJKSTRA ALGORITHM
  - ALL LABELS OF NODES ARE FINAL
  - RUNS IN $O(n^2)$ OR $O(m + n \log n)$ WITH PRIORITY QUEUE

- BELLMAN-FORD ALGORITHM

$\ell(s) = 0$
$\ell(v) = \infty$ FOR $v \neq S$

FOR $i = 1$ TO $n-1$:
  FOR EVERY EDGE OF GRAPH $(v,w) \in E(G)$:
    IF $\ell(w) > \ell(v) + c(v,w)$:
      $\ell(w) = \ell(v) + c(v,w)$
      $p(w) = v$

- IF MAIN LOOP DOESN'T DO ANY CHANGE, YOU CAN TERMINATE THE ALGORITHM
- IT IS BEST KNOWN ALGORITHM TO FIND SHORTEST PATH TREE
  - COMPLEXITY $O(nm)$
- IT CAN DETECT NEGATIVE CYCLE

- FLOYD ALGORITHM

- INPUT IS GRAPH WITHOUT CYCLES OF NEGATIVE WEIGHT

- $c : E(G) \to \mathbb{R}$
- OUTPUT : MATRIX l_{ij} WITH SHORTEST PATH LENGTHS, P_{ij} - LAST BUT ONE NODE ON PATH

$l_{ij} = c((i,j))$ FOR ALL $(i,j) \in E(G)$

$l_{ij} = \infty$ FOR ALL $(i,j) \in E(G)$ WHERE $i \neq j$

$l_{ii} = 0$ FOR ALL i

$P_{i,j} = i$ FOR ALL (i,j)

FOR $k = 1$ TO n:

FOR $i = 1$ TO n:

FOR $j = 1$ TO n:

IF $l_{ij} > l_{ik} + l_{kj}$ THEN:

$l_{ij} = l_{ik} + l_{kj}$

$P_{ij} = P_{kj}$

- BEST KNOW ALGORITHM FOR ALL PAIRS SHORTEST PATH WITHOUT NEGATIVE CYCLES

- TIME COMPLEXITY IS $O(n^3)$

- THERE IS NEGATIVE CYCLE IF THERE EXISTS i SUCH THAT $l_{ii} < 0$

- CAN BE USED TO FIND MINIMAL WEIGH CYCLE

- IT WILL APPEAR ON DIAGONAL

- SHORTEST PATH IN DAG

  - TOPOLOGICAL ORDER

    - ORDER OF VERTICES SUCH THAT FOR EACH EDGE $(v_i, v_j) \in E(G)$ WE HAVE $i < j$

- OBSERVATION

  - SHORTEST PATH FROM s TO v_i CANNOT USE ANY NODE FROM $v_{i+1, \dots, n}$

- SIMPLIFIED BELLMAN FORD

  $\ell(v_1) = 0$

  $\ell(v_i) = \infty$ FOR $i = 2 \dots n$

  FOR $i = 2$ TO n:

    FOR EVERY EDGE OF GRAPH $(v_j, v_i) \in E(G)$:

      IF $\ell(v_i) > \ell(v_j) + c(v_j, w)$: # DIFFERENCES

        $\ell(v_i) = \ell(v_j) + c(v_j, v_i)$

        $P(v_i) = v_j$

- WE DONT HAVE TO CHECK EVERY EDGE

- TIME COMPLEXITY $O(|V| + |E|)$

- "STARTS WITH SIMPLE PROBLEMS AND GRADUALLY SOLVES LARGER PROBLEMS ALONG TOPOLOGICAL ORDER"

  - BASICALLY DYNAMIC PROGRAMMING

- NETWORK
  - 5-TOUPLE (G, ℓ, u, s, t)
    - G - ORIENTED GRAPH
    - $u: E(G) \to \mathbb{R}_0^+$ - MAXIMUM CAPACITY OF ARC
    - $\ell: E(G) \to \mathbb{R}_0^+$ - MINIMUM CAPACITY OF ARC
    - s - SOURCE NODE
    - t - SINK NODE

- NETWORK FLOW
  - $f: E(G) \to \mathbb{R}_0^+$
    - IT IS KIRCHHOFF LAW IF $\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$ IS VALID FOR ALL NODES EXCEPT s AND t (↖ SET OF ARC ENTERING, ↖ SET OF ARC LEAVING)

- FEASIBLE FLOW
  - MUST SATISFY $f(e) \in \langle \ell(e), u(e) \rangle$
  - THERE MAY BE NO FEASIBLE FLOW WHEN $\ell(e) > 0$

- MAXIMAL FLOW
  - GIVEN NETWORK (G, ℓ, u, s, t)
  - GOAL IS TO FIND FEASIBLE FLOW
  - THAT MAXIMIZES $\sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$
  - "SEND MAXIMUM FROM s TO t"

- MAXIMUM FLOW AS LP

  - VARIABLE $f(e) \in \mathbb{R}_0^+$ REPRESENTS FLOW THROUGH ARC $e \in E(G)$

$$\text{MAX} \sum_{e \in \sigma^+(s)} f(e) - \sum_{e \in \sigma^-(s)} f(e)$$

$$\text{SE} \sum_{e \in \sigma^-(v)} f(e) = \sum_{e \in \sigma^+(v)} f(e) \qquad v \in V(G) \setminus \{s, t\}$$

$$\ell(e) \leq f(e) \leq u(e) \qquad e \in E(G)$$

- FORD - FULKERSON ALGORITHM

  - ALGORITHM TO FIND MAXIMAL FLOW

  - INCREMENTAL AUGMENTATION OF FLOW ALONG PATH FROM SOURCE s TO SINK t WHILE MAINTAINING FLOW'S FEASIBILITY

  - PATH DOESN'T RESPECT ORIENTATION OF EDGES

  - AUGMENTING PATH

    - AUGMENTING PATH FOR FLOW f IS A PATH FROM s TO t

      - $f(e) < u(e)$ IF e IS FORWARD EDGE

        - FLOW CAN BE INCREASED

      - $f(e) > \ell(e)$ IF e IS BACKWARD EDGE

        - FLOW CAN BE DECREASED

  - CAPACITY OF AUGMENTING PATH

    - BIGGEST POSSIBLE INCREASE OF THE FLOW ON IT

- 1. FIND FEASIBLE FLOW $f(e)$ FOR ALL $e \in E(G)$
- 2. FIND AUGMENTING PATH P. IF NON EXISTS STOP
- 3. COMPUTE CAPACITY γ OF P, AUGMENT THE FLOW FROM s TO t AND GO TO 2
    - INCREASE FLOW BY γ ON FORWARD EDGES AND DECREASE BY γ ON BACKWARD EDGES
    - THIS AUGMENTING PATH CAN'T BE USED AGAIN BECAUSE FLOW ON IT IS HIGHEST POSSIBLE
    - FLOW IS MAXIMUM ONLY IF THERE IS NO AUGMENTING PATH

- CUT
    - IN G
    - SET OF EDGES $\delta(A)$ WITH $s \in A$ AND $t \in V(G) \setminus A$
    - "CUT SEPARATES s AND t"
    - MINIMUM CUT
        - CUT OF MINIMAL CAPACITY

$$c(A) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$$

- VALUE OF MAXIMUM FLOW FROM s TO t IS EQUAL TO THE CAPACITY OF MINIMUM CUT
    - THIS FLOWS FROM LP DUALITY
-

- INTEGRAL FLOW THEOREM
  - IF CAPACITIES OF NETWORKS ARE INTEGERS, THAN INTEGER-VALUED MAXIMUM FLOW EXISTS
  - IF WE CHOOSE AUGMENTING PATH INAPPROPRIATELY WE CAN AUGMENT FLOW BY UNIT STEPS ONLY
    - IF THERE ARE NON-INTEGER CAPACITIES AND NON-INTEGER FLOW THE ALGORITHM MIGHT NOT TERMINATE AT ALL
  - WHEN WE CHOOSE SHORTEST AUGMENTING PATH FIRST
    - TIME COMPLEXITY IS $O(m^2 \cdot n)$

- FEASIBLE FLOW WITH BALANCES
  - WE ASSUME SEVERAL SOURCES AND SINKS
  - NO LOWER BOUNDS
  - IT CAN BE POLYNOMIALLY REDUCED TO MAXIMUM FLOW PROBLEM
    - WITH ONE SOURCE AND SINK
  - LET (G, w, b) BE A NETWORK WHERE
    - G IS GRAPH
    - w IS UPPER BOUND
    - b IS BALANCE
      - $b: V(G) \to \mathbb{R}$
      - REPRESENTS SUPPLY OR CONSUMPTION OF NODES
      - AND SATISFIES $\sum_{v \in V(G)} b(v) = 0$
    - GOAL IS TO DECIDE IF THERE EXISTS FEASIBLE FLOW f THAT SATISFIES

$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v) \quad \text{FOR ALL } v$$

- WE CAN TRANSFORM THIS PROBLEM TO MAXIMUM FLOW PROBLEM WITH ZERO LOWER BOUNDS
  - WE ADD NEW SOURCE NODE s AND ADD EDGES WITH UPPER BOUND $u_v = b(v)$ FOR EVERY NODE SATISFIING $b(v) > 0$
  - WE ADD NEW SINK NODE t AND ADD EDGES WITH UPPER BOUND $u_v = -b(v)$ FOR EVERY NODE SATISFIING $b(v) < 0$
  - WE SOLVE MAXIMUM FLOW PROBLEM FOR LOWER BOUNDS EQUAL TO ZERO
  - IF MAXIMUM FLOW SATURATES ALL EDGES LEAVING s AND/OR ENTERING t THE FEASIBLE FLOW EXISTS

- INITIAL FLOW FOR FORD-FULKERSON ALGORITHM
  - IF ALL EDGES HAVE LOWER BOUND 0, WE USE LOWER BOUND AS INITIAL FLOW
  - ELSE WE TRANSFORM THE PROBLEM TO FEASIBLE FLOW DECISION PROBLEM
    - WE ADD ARC FROM t TO s WITH INFINITE CAPACITY
      - SO KIRHOFF'S LAW APPLIES TO s AND t NOW TOO
    - SUBSTITUTE $f(e) = f(e)' + l(e)$ AND WE OBTAIN TRANSFORMED PROBLEM

$$0 \le f(e)' \le u(e) - l(e)$$

$$\sum_{e \in \delta^+(v)} f(e)' - \sum_{e \in \delta^-(v)} f(e)' = \underbrace{\sum_{e \in \delta^-(v)} l(e) - \sum_{e \in \delta^+(v)} l(e)}_{b(v)}$$

- THIS IS FEASIBLE FLOW DECISION PROBLEM BECAUSE

$$\sum_{v \in V(G)} b(v) = 0$$

- WHILE SOLVING IT WE OBTAIN THE INITIAL FEASIBLE FLOW OR DECIDE IT DOESN'T EXISTS

- MINIMUM COST FLOW

- 5-TOUPLE (G, l, u, c, b)

G - GRAPH

l - LOWER BOUND

u - UPPER BOUND

c - COST OF ARC

b - BALANCE

- GOAL IS TO FIND FEASIBLE FLOW f THAT MINIMIZES $\sum_{e \in E(G)} f(e) \cdot c(e)$ AND SATISFIES $\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$ FOR ALL v

- LP FORMULATION

$$\min \sum_{e \in E(G)} c(e) \cdot f(e)$$

$$\text{s.t.} \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$$

$$l(e) \leq f(e) \leq u(e)$$

- MAXIMUM FLOW PROBLEM CAN BE TRANSFORMED TO MINIMUM COST FLOW

- ADD EDGE FROM t TO s WITH UPPER BOUND EQUAL TO ∞ AND COST -1

- EVERY OTHE COST IS 0

- SET $b(v) = 0$ FOR ALL NODES INCLUDING s AND t

- MATCHING
  - SET OF EDGES $P \subseteq E(G)$
  - SUCH THAT ENDPOINTS ARE ALL DIFFERENT
  - IF ALL NODES ARE INCIDENT WITH SOME EDGE IN P WE CALL IT PERFECT MATCHING
  - PROBLEMS
    - MAXIMUM CARDINALITY MATCHING PROBLEM
      - MATCHING WITH MAXIMUM NUMBER OF EDGES
    - MINIMUM WEIGHT MATCHING
      - FIND CHEAPEST MATCHING FROM SET OF ALL MAXIMUM CARDINALITY MATCHINGS
    - MINIMUM WEIGH PERFECT MATCHING
  - IT CAN BE SOLVED IN POLYNOMIAL TIME

- MAXIMUM CARDINALITY MATCHING PROBLEM
  - M- ALTERNATING PATH P
    - IF $E(P) \setminus M$ IS MATCHING
  - M- AUGMENTING
    - IF ENDPOINTS ARE NOT COVERED BY M

  - MATCHING M IS MAXIMUM IF THERE IS NO M-AUGMENTING PATH
  - ALGORITHM
    - FIND ANY MATCHING M OF GRAPH G
    - FIND M-AUGMENTING PATH AND CHANGE ITS COVERAGE

- HUNGARIAN ALGORITHM

  - SOLVES ASSIGNMENT PROBLEM

  - n WORKERS, n JOBS

  - MATRIX OF COST

$$\begin{bmatrix} 30 & 25 & 10 \\ 15 & 10 & 20 \\ 25 & 20 & 15 \end{bmatrix}$$

- 1. SUBSTRACT MIN OF ROWS

$$\begin{bmatrix} 20 & 15 & 0 \\ 5 & 0 & 10 \\ 10 & 5 & 0 \end{bmatrix} \begin{matrix} 10 \\ 10 \\ 15 \end{matrix}$$

- 2. SUBSTRACT MIN OF COLUMNS

$$\begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix}$$

$$\begin{matrix} 5 & 0 & 0 \end{matrix}$$

- 3. COVER ALL ZEROS WITH LOWEST POSSIBLE NUMBER OF STRAIGHT LINES

$$\begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix}$$

2 LINES < 3 WORKERS

- GO TO 4.

4. FIND MIN OF POSITIONS NOT COVERED BY LINE

SUBSTRACT MIN FROM NOT-COVERED POSITIONS

ADD MIN TO ALL LINE INTERSECTIONS

$$\left[\begin{array}{ccc|c} 15 & 15 & 0 & \\ 0 & 0 & 10 & \\ 5 & 5 & 0 & \end{array}\right] \quad 5$$

$$\left[\begin{array}{ccc|c} 10 & 10 & 0 & \\ 0 & 0 & 15 & \\ 0 & 0 & 0 & \end{array}\right]$$

3 LINES = 3 WORKERS

GO TO 5.

5. MAKE FINAL ASSIGNMENT

$$\begin{bmatrix} 10 & 10 & \textcircled{0} \\ \textcircled{0} & 0 & 15 \\ 0 & \textcircled{0} & 0 \end{bmatrix}$$

- MULTICOMODITY FLOW PROBLEM

  - EACH COMODIDY HAS SEVERAL SOURCES AND SINKS

  - $f^m(e)$ IS FLOW OF COMODITY $m \in M$ ALONG EDGE e

  - MINIMUM COST MULTICOMODITY FLOW PROBLEM

    - 5 -TOUPLE $(G, l, u, c, b^1 \dots b^m \dots b^{|M|})$

      - VECTOR $b^m : V(G) \to \mathbb{R}$

        - SUPPLY OR CONSUMPTION OF NODES BY COMMODITY m.

        - $\sum_{v \in V(G)} b^m(v) = 0$ FOR ALL COMMODITIES $m \in M$

    - GOAL IS TO FIND FEASIBLE FLOW f WHOSE COST

      $\sum_{e \in E(G)} \sum_{m \in M} f^m(e) \cdot c(e)$ IS MINIMAL

    - FEASIBLE FLOW SATISFIES $\sum_{e \in \delta^+(v)} f^m(e) - \sum_{e \in \delta^-(v)} f^m(e) = b^m(v)$

      FOR ALL v AND ALL m

  - LP FORMULATION

    $$\min \sum_{e \in E(G)} \sum_{m \in M} f^m(e) \cdot c(e)$$

    $$\text{s.t.} \sum_{e \in \delta^+(v)} f^m(e) - \sum_{e \in \delta^-(v)} f^m(e) = b^m(v)$$

    $$l(e) \le \sum_{m \in M} f^m(e) \le u(e)$$

- KNAPSACK PROBLEM

  - n - NUMBER OF ITEMS

  - $c_1, \dots c_n$ - COST OF EACH ITEM

  - $w_1, \dots, w_n$ - WEIGHT OF EACH ITEM

  - W - MAXIMUM WEIGHT OF KNAPSACK

  - GOAL IS TO FIND SUBSET $S \subseteq \{1, \dots, n\}$ SUCH THAT

    $$\sum_{j \in S} w_j \leq W \quad \text{AND} \quad \sum_{j \in S} c_j \text{ IS MAXIMUM}$$

  - ONE OF EASIEST NP-HARD PROBLEMS

  - ALSO CALLED 0-1 KNAPSACK

- FRACTIONAL KNAPSACK PROBLEM

  - WE CAN DIVIDE ITEMS

  - FIND RATIONAL NUMBERS $x_1, \dots, x_j, \dots x_n$ SUCH THAT

    $$0 \leq x_j \leq 1 \quad \text{AND} \quad \sum_{j=1}^{n} x_j \cdot w_j \leq W \quad \text{AND} \quad \sum_{j=1}^{n} x_j c_j \text{ IS MAXIMUM}$$

  - THIS CAN BE SOLVED IN POLYNOMIAL TIME

    - SORT ITEMS BY $\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}$

    - FIND THE FIRST ITEM WHICH DOESN'T FIT

      $$k := \min \left\{ j \in \{1, \dots, n\} : \sum_{i=1}^{j} w_i > W \right\}$$

    - CUT PART OF ITEM WHICH DOESN'T FIT

      $$x_k = \frac{W - \sum_{i=1}^{k-1} w_i}{w_k}$$

    - SORTING TAKES $O(n \log n)$, COMPUTATION OF k IS $O(n)$

  - TOTAL TIME IS $O(n \log n)$

- 2 - APPROXIMATION ALGORITHM FOR KNAPSACK

- r - APPROXIMATION ALGORITHM FOR MAXIMALIZATION

- $r \geq 1$
- $J^A(I) \geq \frac{1}{r} J^*(I)$
  - J^A: APPROXIMATION ALGORITHM
  - J^*: OPTIMAL SOLUTION
- " THE SOLUTION OF APPROXIMATING ALGORITHM IS AT LEAST BIGGER THAN $\frac{1}{r}$ OF OPTIMAL SOLUTION"

- SAME AS FRACTIONAL KNAPSACK PROBLEM

- SORT BY $\frac{c_1}{w_1} \geq \dots \geq \frac{c_n}{w_n}$
- FIND ITEM k WHICH DOESN'T FIT
- CHOOSE BETTER OF TWO SOLUTIONS $\{1, \dots, k-1\}$ OR $\{k\}$
- TIME COMPLEXITY $O(n)$
- SINCE $\sum_{i=1}^{k} c_i$ IS UPPER BOUND OF OPTIMUM VALUE

  THE BETTER OF TWO SOLUTIONS $\{1, \dots, k-1\}$ AND $\{k\}$

  ACHIEVES AT LEAST HALF OF OPTIMUM VALUE

- DYNAMIC PROGRAMMING
  - PSEUDO POLYNOMIAL ALGORITHM
  - COMPLEXITY $O(n \cdot C)$
  - USES BELLMAN'S PRINCIPLE OF OPTIMALITY
  - HAS OVERLAPPING SUBPROBLEMS
  - COMPLEXITY REDUCTION
    - DIVIDE ALL COSTS BY 2 AND ROUND THEM DOWN
    - ALGORITHM IS NOW FASTER, BUT CAN RETURN SUBOPTIMAL SOLUTION
    - THIS ALLOWS US TO FIND TRADEOFF BETWEEN SPEED AND DESIRED OPTIMALITY

- $W = 7 \quad c = [1, 4, 5, 7] \quad w = [1, 3, 4, 5]$

c	w	0	1	2	3	4	5	6	7
1	1	0	1	1	1	1	1	1	1
4	3	0	1	1	4	5	5	5	5
5	4	0	1	1	1	5	6	6	9
7	5	0	1	1	1	1	7	8	9

MAX c

- ~~FOR EXCEL~~

$$[x][y] = MAX\{[x][y-1]; [x-w][x-1]+c\}$$

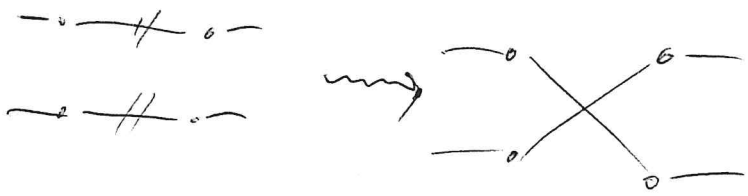
I_1 I_2

- TSP
  - TRAVELING SALESMAN PROBLEM
    - RELATED PROBLEM
      - EXISTENCE OF HAMILTONIAN CYRCUIT
        - DOES CIRCUIT VISITING EVERY NODE EXACTLY ONCE EXISTS IN G
      - NP- COMPLETE
    - COMPLETE UNDIRECTED GRAPH $K_n (n \geq 3)$ AND WEIGHTS $c : E(k_n) \rightarrow Q_0^+$
    - FIND HAMILTONIAN CIRCUIT T WHOSE WEIGHT $\sum_{e \in E(T)} c(e)$ IS MINIMUM
    - IF COST $(A,B) = (B,A)$ THAN WE HAVE SYMETRIC TSP
    - STRONGLY NP- HARD PROBLEMS CAN'T BE SOLVED BY PSEUDOPOLYNOMIAL ALGORITHM UNLESS P=NP
    - TSP IS STRONGLY NP-HARD
    - THERE IS ALSO NO r- APROXIMATION ALGORITHM FOR TSP WORKING IN POLYNOMIAL TIME

- METRIC TSP
  - $c(\{i,j\}) + c(\{j,k\}) \geq c(\{k,i\})$ FOR ALL $i,j,k \in V$
  - FIND HAMILTONIAN CIRCUIT T SUCH THAT $\sum_{e \in E(T)} c(e)$ IS MINIMAL
  - IS NP-HARD STRONGLY
  - BUT APPROXIMATION ALGORITHMS DO EXIST

- THERE ARE ALSO HEURISTICS
  - NEAREST NEIGHBOR
  - SELECT ARBITRARY V AND CONNECT IT TO CLOSEST V'
  - IT RUNS IN $O(n^2)$
  - DOUBLE-TREE ALGORITHM
    - FIND MST
    - DOUBLE EDGES
    - SKIP NODES VISITED TWO TIMES
    - $O(n^2)$
    - IT IS 2-APPROXIMATION ALGORITHM
- CHRISTOFIDES ALGORITHM
  - FIND MST
  - SELECT VERTICES OF MST WITH ODD DEGREE
  - FIND MINIMUM WEIGHT MATCHING OF W IN ORIGINAL GRAPH
  - MERGE MINIMUM WEIGHT MATCHING GRAPH AND MST
  - CREATE EULERIAN WALK
  - TRANSFORM IT TO # HAMILTONIAN CIRCUIT
  - $O(n^3)$
  - $\frac{3}{2}$ APPROXIMATION ALGORITHM

- k-OPT LOCAL SEARCH
  - ONE OF MOST SUCCESSFULL TECHNIQUE IN PRACTISE
- TSP - FIND ANY HAMILTONIAN CIRCUIT
  - IMPROVE IT BY LOCAL MODIFICATIONS
    - FOR EXAMPLE DELETE TWO EDGES AND RECONSTRUCT CIRCUIT BY OTHER EDGES



- CSP
  - CONSTRAINT (SATISFACTORY) PROGRAMMING
  - DECLARATIVE PROGRAMMING
    - MODEL : VARIABLES, DOMAINS, CONSTRAINTS
    - SOLVER : PROPAGATION, SEARCHING
  - DEFINED BY TRIPLET
    - (X, D, C)
      - X IS FINITE SET OF VARIABLES $\{x_1, \dots, x_m\}$
      - $D = \{D_1, \dots, D_m\}$ IS FINITE SET OF DOMAINS OF VARIABLES
      - $C = \{c_1, \dots, c_e\}$ IS FINITE SET OF CONSTRAINTS
    - DOMAIN $D_i = \{v_1, \dots, v_2\}$ IS A FINITE SET OF ALL POSSIBLE VALUES OF x_i
    - CONSTRAINT c_i IS A COUPLE (S_i, R_i) WHERE $S_i \subseteq X$ AND R_i IS A RELATION OVER THE SET OF VARIABLES S_i. FOR $S_i = \{x_{i_1}, \dots x_{i_r}\}$ IS $R_i \subseteq D_{i_1} \times \dots \times$

- IT IS NP-COMPLETE PROBLEM

- SOLUTION IS COMPLETE ASSIGNMENT OF VALUES FROM DOMAINS TO VARIABLES SUCH THAT ALL CONSTRAINTS ARE SATISFIED

  - THIS IS ONLY DECISION PROBLEM

- CONSTRAINT SATISFACTION OPTIMIZATION PROBLEM

  - $(X, D, C, f(X))$

    - $f(X)$ IS OBJECTIVE FUNCTION

    - THE SEARCH IS NOT STOPPED WHEN THE FIRST SOLUTION SATISFYING ALL CONSTRAINTS IS FOUNDED.

    - IT IS STOPPED WHEN OPTIMAL SOLUTION IS FOUNDED

- CONSTRAINT SOLVING

  - DEFINED BY (X, D, C) WHERE D_i IS DEFINED ON R

    - IT SOLVES SET OF LINEAR EQUATIONS \ MINOR EQUATIONS

- COMPARISION OF CSP WITH ILP

  - BOTH DECLARATIVE PROGRAMING

  - CSP ALLOWS TO FORMULATE COMPLEX CONSTRAINTS

  - ILP USES INEQUALITIES ONLY

  - CSP CAN USE ARBITRARY RELATIONS

  - IT IS DIFFICULT TO REPRESENT CONTINUOUS PROBLEMS BY CSP

    - BUT WE CAN USE HYBRID APPROACHES AND COMBINE IT WITH LP

- ARC CONSISTENCY

  - WE CONSIDER BINARY CSP ONLY

    - ALL CSP CAN BE CONVERTED TO BINARY CSP

  - BINARY CSP CAN BE REPRESENTED BY GRAPH

    - NODES ARE VARIABLES

      - IF THERE IS CONSTRAINT INVOLVING X_i, X_j THEN THE NODES ARE CONNECTED IN BOTH WAYS

  - ARC (X_i, X_j) IS ARC CONSISTENT IFF

    - FOR EACH VALUE $a \in D_i$ THERE EXISTS VALUE $b \in D_j$ SUCH THAT ASSIGNMENT $X_i = a$, $X_j = b$ MEETS ALL BINARY CONSTRAINTS FOR VARIABLES X_i, X_j

  - CSP IS ARC CONSISTENT IF ALL ARC ARE ARC CONSISTENT

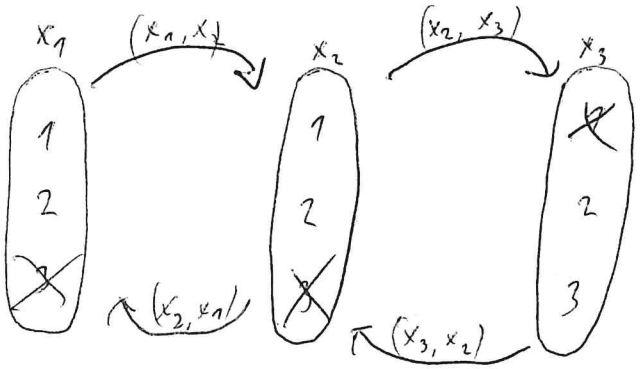
  - AC IS ORIENTED

    - CONSISTENCE OF (X_i, X_j) DOESN'T GUARANTEE CONSISTENCE OF (X_j, X_i)

- AC-3 EXAMPLE

$X = \{x_1, x_2, x_3\}$ $\quad x_1 = x_2 \quad x_2 + 1 = x_3$

$D_1 = \{1, 2, 3\}$ $\quad D_2 = \{1, 2, 3\}$ $\quad D_3 = \{1, 2, 3\}$



Queue $Q = \{(x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2)\}$

1. (x_1, x_2) ✓

   $Q = \{(x_2, x_1), (x_2, x_3), (x_3, x_2)\}$

2. (x_2, x_1) ✓

   $Q \{(x_2, x_3), (x_3, x_2)\}$

3. (x_2, x_3) X

   $x_2 + 1 = x_3$

   $x_2 = 3$ X

   $Q \{(x_3, x_2), (x_1, x_2)\}$

4. (x_3, x_2) X

   $x_2 + 1 = x_3$

   $x_3 = 1$ X

   $Q \{(x_1, x_2), (x_2, x_3)\}$

5. (x_1, x_2) X

   $x_1 = x_2$

   $x_1 = 3$ X

   $Q \{(x_2, x_3), (x_2, x_1)\}$

6. (x_2, x_3) ✓

   $Q \{(x_2, x_1)\}$

7. (x_2, x_1) ✓

   $Q = \emptyset$

END.

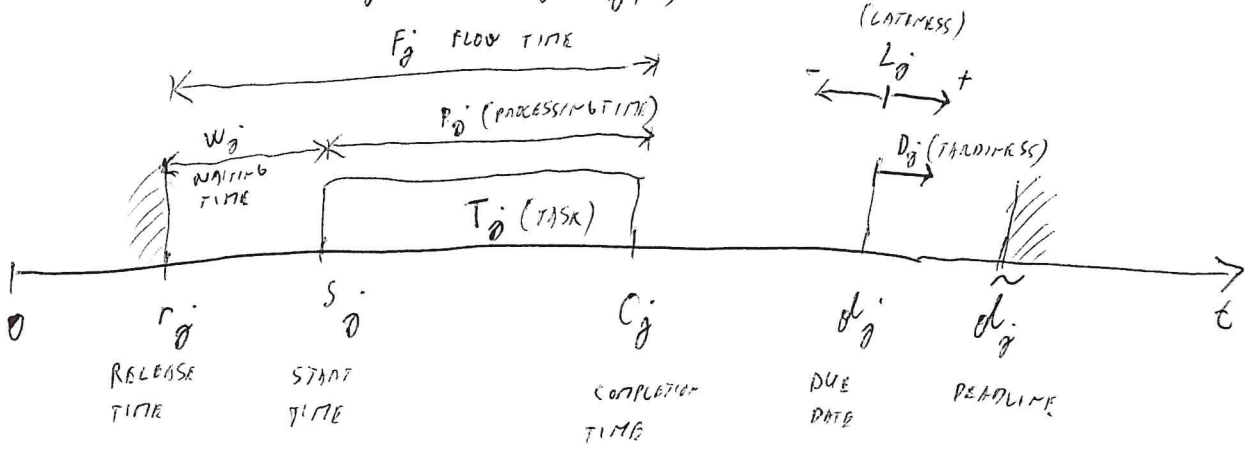
- SCHEDULING

- SET OF n TASKS $T = \{T_1, T_2, \dots T_n\}$
- SET OF m RESOURCES WITH CAPACITIES R_K

$$P = \{P_1^1, \dots, P_1^{R_1}, P_2^1, \dots, P_2^{R_2}, \dots, P_m^1, \dots, P_m^{R_m}\}$$

- IT IS TASK OF ASSIGNMENT OF TASKS TO RESOURCES IN TIME
- EACH TASK HAS TO BE COMPLETED
  - THIS IS DIFFERENCE TO PLANNING (IT DECIDES WHAT TO DO AND PROCESS)
- OFF-LINE SCHEDULING
  - TASKS ARE KNOWN BEFORE
- ON-LINE SCHEDULING
  - NEW TASKS APPEARS
- THE RESULT OF SCHEDULING CAN BE DEPICTED AS GANTT CHART
- GENERAL CONSTRAINTS
  - EACH TASK CAN BE PROCESSED BY AT MOST ONE RESOURCE AT TIME
  - EACH RESOURCE CAN PROCESS ONLY ONE TASK AT TIME
- SPECIFIC CONSTRAINTS
  - TASK T_i HAS TO BE PROCESSED DURING TIME INTERVAL $\langle r_i, \hat{d}_i \rangle$
  - $T_i \prec T_j$
    - PROCESS OF T_j CAN'T START BEFORE TASK T_i WAS COMPLETED
  - NON-PREEMPTIVE SCHEDULING
    - TASK CAN'T BE STOPPED AND COMPLETED LATER
  - PREEMPTIVE SCHEDULING
    - TASK CAN BE STOPPED
    - BUT NUMBER OF STOPS MUST BE LIMITE

- TASK
  - PARAMETERS
    - RELEASE TIME r_j
    - PROCESSING TIME p_j
    - DUE DATE d_j
      - TIME IN WHICH TASK SHOULD BE COMPLETED
    - DEADLINE $\tilde{d}_j$
      - TIME IN WHICH TASK HAVE TO BE COMPLETED
- VARIABLES
  - START TIME S_j
  - COMPLETION TIME C_j
  - WAITING TIME $W_j = S_j - r_j$
  - FLOW (LEAD) TIME $F_j = C_j - r_j$
  - LATENESS $L_j = C_j - d_j$
  - TARDINESS $D_j = MAX\{C_j - d_j, 0\}$



- GRAHM'S NOTATION
  - $\alpha|\beta|\gamma$
  - RESOURCE / TASK / CRITERION
  - α
    - RESOURCES
      - PARALLEL
        - ONLY ONE TYPE OF RESOURCE
        - SAME CAPACITY R
        - TASK CAN RUN ON ANY RESOURCE
      - DEDICATED
        - TASK CAN RUN ONLY ON ONE RESOURCE
    - PROJECT SCHEDULING
      - m RESOURCE TYPES
      - EACH WITH CAPACITY R_k
  - CHARACTERISTICS α_1, α_2
    - α_1
      - 1 = SINGLE RESOURCE
      - P = PARALLEL IDENTICAL RESOURCES
      - Q = PARALLEL UNIFORM RESOURCES (COMPUTATION TIME IS INVERSELY RELATED TO RESOURCE SPEED)
      - R = PARALLEL UNRELATED RESOURCES (COMPUTATION TIMES ARE GIVEN AS A MATRIX, RESOURCE × TASKS)
      - O = DEDICATED RESOURCES OPEN-SHOP (TASKS ARE INDEPENDENT)
      - F = DEDICATED RESOURCES FLOW-SHOP (TASKS ARE GROUPED IN SEQUENCES IN THE SAME ORDER, EACH JOB VISITS EACH MACHINE ONCE)
      - J = DEDICATED RESOURCES JOB-SHOP (ORDER OF ~~TASKS~~ TASKS IN JOBS IS ARBITRARY, RESOURCES CAN BE USED SEVERAL TIMES IN A JOB)
      - PS = PROJECT SCHEDULING (MOST GENERAL)

- α_2
  - $\emptyset$ - ARBITRARY NUMBER OF RESOURCES
  - 2 - 2 RESOURCES
  - m, R - m RESOURCE TYPES WITH CAPACITIES R (PROJECT SCHEDULING)

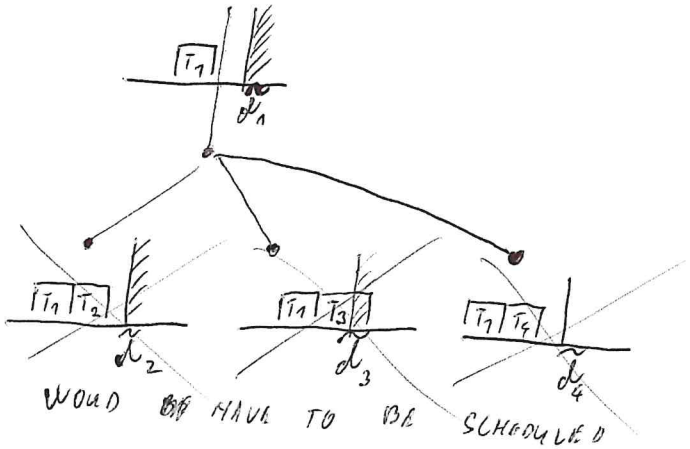
- TASK CHARACTERISTICS $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8$
  - β_1
    - PMTN - PREEMPTION ALLOWED
    - $\emptyset$ - PREEMPTION NOT ALLOWED
  - β_2
    - PREC - PRECEDENCE CONSTRAINTS
    - IN-TREE, OUT-TREE - TREE CONSTRAINTS
    - CHAIN - CHAIN CONSTRAINTS
    - TMP - TEMPORAL CONSTRAINTS (FOR PROJECT SCHEDULING)
    - $\emptyset$ - INDEPENDENT TASKS
  - β_3
    - r_j - RELEASE TIME
  - β_4
    - $p_j = p$ - UNIFORM PROCESSING TIME
    - $p_L \leq p_j \leq p_U$ - RESTRICTED PROCESSING TIME
    - $\emptyset$ - ARBITRARY PROCESSING TIME
  - β_5
    - $\tilde{d}_j, d_j$ - DEADLINE, DUE-DATE
  - β_6
    - $m_j \leq k$ - MAXIMAL NUMBER OF TASKS IN A JOB

- β_7
  - NO-WAIT - BUFFERS OF ZERO CAPACITY
- β_8
  - SET-UP - TIME FOR RESOURCE RECONFIGURATION

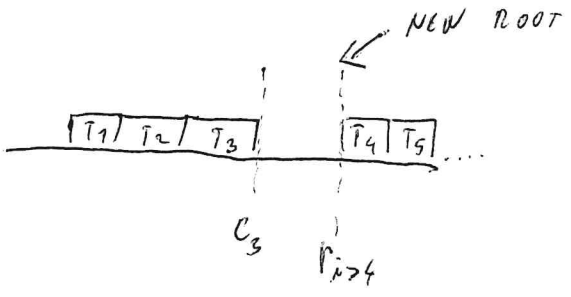
- OPTIMAL CRITERION γ
  - γ
    - C_{MAX}
      - MINIMIZE SCHEDULE LENGTH (COMPLETION TIME OF LAST TASK)
    - $\sum C_j$
      - MINIMIZE SUM OF COMPLETION TIMES
    - $\sum w_j C_j$
      - MINIMIZE WEIGHTED COMPLETION TIME
    - L_{MAX}
      - MINIMIZE LATENESS
    - $\emptyset$ - DECISION PROBLEM

- SCHEDULING ON ONE RESOURCE
  - $1|prec|C_{MAX}$
    - EASY
    - TASKS ARE PROCESSED IN ARBITRARY ORDER THAT SATISFIES PRECEDENCE RELATION
  - $1||C_{MAX}$
    - EASY
  - $1|r_j|C_{MAX}$
    - EASY
    - SOLVE TASKS WITH LOWEST r_j FIRST
  - $1|\tilde{d}_j|C_{MAX}$
    - FAST
    - SOLVE BY EARLIEST DEADLINE FIRST
    - SOLUTION DOESN'T HAVE TO EXIST

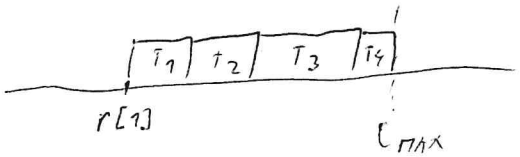
- $1|r_j, \tilde{d}_j|C_{MAX}$
  - NP-HARD
  - FOR TASKS OF UNIT LENGTH POLYNOMIAL ALGORITHM EXISTS
    - BRATLEY'S ALGORITHM
      - BRANCH AND BOUND
      - CREATES SOLUTION TREE
      - REDUCING THE TREE
        - ELIMINATE NODE EXCEEDING DEADLINE AND ITS BROTHER



        - T_3 WOULD HAVE TO BE SCHEDULED ANYWAY IN EACH BRANCHES
    - DECOMPOSITION
      - WHEN ALL RELEASED TASKS ARE SCHEDULED AND C_i IS LOWER THAN r_N OF ANY NON RELEASED TASKS, THE SCHEDULED TASKS ARE OPTIMAL
      - WE DONT HAVE TO BACKTRACK



- OPTIMALITY TEST
  - IF FIRST TASK STARTS AT ITS RELEASE TIME, AND RELEASE TIMES ARE GROWING WITH EACH TASK
  - AND ALL TASKS RUNS WITHOAT IDLE WAITING
  - THAN THERE IS BRTP
    - BLOCK WITH RELEASE TIME PROPERTY
  - IF BRTP EXISTS, THAN SCHEDULE IS OPTIMAL



  - BUT IT CAN BE OPTIMAL EVEN WITHOUT BRTP

- SCHEDULING ON ONE RESOURCE MINIMIZING $\sum w_j C_j$
  - $1||\sum C_j$
    - EASY
    - SHORTEST PROCESSING TIME FIRST
  - $1||\sum w_j C_j$
    - EASY
    - WEIGHTED SHORTEST PROCESSING TIME FIRST
    - SCHEDULE TASKS ACCORDING TO $\frac{p_j}{w_j}$
  - $1|r_j|\sum C_j$
    - NP-HARD
  - $1|PMTN, r_j|\sum C_j$
    - SOLVED BY MODIFIED SPT
  - $1|PMTN, r_j|\sum w_j C_j$
    - NP-HARD

- $1|\tilde{d}_j|\sum C_j$
  - SOLVABLE BY MODIFIED SPT
- $1|\tilde{d}_j|\sum w_j C_j$
  - NP-HARD
- $1|prec|\sum C_j$
  - NP-HARD

- BRANCH AND BOUND WITH LP FOR $1|prec|\sum w_j C_j$
  - FIRST FORMULATE PROBLEM AS ILP
    - VARIABLE $x_{ij} \in \{0,1\}$
      - IF T_i PRECEDES T_j OR $i=j$
        - $x_{ij}=1$
    - ENCODING OF PRECEDENCE RELATIONS
      - $l_{ij} \in \{0,1\}$
      - $l_{ij}=1$ IF THERE IS EDGE FROM T_i TO T_j IN THE PRECEDENCE GRAPH G OR $i=j$
    - CRITERION
      - COMPLETION TIME OF TASK T_j CONSISTS OF p_j AND PROCESSING TIME OF ITS PREDECESSORS
      - $J = \sum_{j=1}^{n} \sum_{i=1}^{n} p_i x_{ij} w_j$
      - FROM ALL FEASIBLE SCHEDULES X WE LOOK FOR ONE THAT MINIMIZES $J(X)$

- FORMAL ILP

$$\text{MIN} \sum_{j=1}^{n} \sum_{i=1}^{n} p_i \cdot x_{ij} \cdot w_j$$

$x_{ij} \geq l_{ij} \quad i,j \in 1..n$ - "IF T_i PRECEDES T_j IN G, THEN IT PRECEDES T_j IN THE SCHEDULE"

$x_{ij} + x_{ji} = 1 \quad i,j \in 1..n \quad i \neq j$ - "EITHER T_i PRECEDES T_j OR VICE VERSA"

$1 \leq x_{ij} + x_{jk} + x_{ki} \leq 2 \quad i,j,k \in 1..n \quad i \neq j \neq k$ - "NO CYCLE EXISTS IN THE DIGRAPH OF X"

$x_{i,i} = 1 \quad i \in 1..n$

PARAMETER : $w_{i \in 1..n}, p_{i \in 1..n} \in \mathbb{R}_0^+ \quad l_{i \in 1..n, j \in 1..n} \in \{0,1\}$

VARIABLES: $x_{i \in 1..n, j \in 1..n} \in \{0,1\}$

- WE RELAX ON THE INTEGRALITY OF VARIABLE X
  - $0 \leq x_{ij} \leq 1$ AND $x_{i \in 1..n, j \in 1..n} \in \mathbb{R}$
- THIS DOESN'T GIVE THE RIGHT SOLUTION
- BUT WE CAN USE J^{LP} (REMAINING TASKS) VALUE AS A LOWER BOUND ON "AMOUNT OF REMAINING WORK"
- BRANCH AND BOUND CREATES TREE
- J_1 IS THE BEST KNOWN SOLUTION SO FAR
  - WE CAN DISCARD PARTIAL SOLUTION WHEN
    - $J_2 \geq J_1$
    - OR $J_2 + J^{LP} \geq J_1$

- SCHEDULING ONE ONE RESOURCE MINIMIZING L_{MAX}

  - $1||L_{MAX}$

    - SOLVED BY ERLIEST DUE DATE FIRST (EDD)

    - RUNS IN POLYNOMIAL TIME

  - $1|r_j|L_{MAX}$

    - NP-HARD

  - $1|r_j, P_j = 1|L_{MAX}$

    - POLYNOMIAL

    - ITERATING EDD

  - $1|PMTN, r_j|L_{MAX}$

    - POLYNOMIAL

    - ITERATING EDD BY HORN

  - $1|PMTN, r_j, d_j = \tilde{d}_j|L_{MAX}$

    - POLYNOMIAL

    - HORN'S ALGORITHM (ALSO CALLED EDF)

  - $1|PMTN, PREC, r_j, d_j = \tilde{d}_j|L_{MAX}$

    - POLYNOMIAL

    - TRANSFORMATION TO INDEPENDENT TASK SET AND THEN EDF

- MINIMIZATION OF L_{MAX} REQUIRES DUE-DATE d_j EVEN IF IT IS NOT STATED IN β

- SCHEDULING ON PARALLEL IDENTICAL RESOURCES MINIMIZING C_{MAX}

- $P2||C_{MAX}$
  - NP-HARD
    - BECAUSE OF 2 PARTITION PROBLEM

- $P|PMTN|C_{MAX}$
  - EASY
  - SOLVED BY MC NAUGHTON ALGORITHM IN $O(n)$

- $P|PMTN, r_j, \tilde{d}_j|-$
  - EASY
  - DECISION VERSION OF MAXIMUM FLOW PROBLEM

- $P|PREC|C_{MAX}$
  - NP-HARD
  - LS-APPROXIMATION ALGORITHM

- $P||C_{MAX}$
  - NP-HARD
  - LPT - APPROXIMATION ALGORITHM
  - DYNAMIC PROGRAMMING

- $P|PMTN, PREC|C_{MAX}$
  - NP-HARD
  - MUNTZ & COFFMAN'S LEVEL ALGORITHM

- LIST SCHEDULING
  - APPROXIMATION ALGORITHM FOR $P|PREC|C_{MAX}$
  - WE HAVE A LIST OF TASKS
  - IF WE HAVE SOME RESOURCE FREE, WE ASSIGN TO IT THE FIRST FREE TASK FROM LIST
  - ACCURACY DEPENDS ON CRITERION AND SORTING PROCEDURE
  - IT ALSO EXHIBITS ANOMALIES
    - C_{MAX} INCREASES IF WE RELAXE SOME CONSTRAINTS
      - DECREASE OF PROCESSING TIME p_i
      - REMOVAL OF SOME PRECEDENCE CONSTRAINTS
      - INCREASE OF THE NUMBER OF RESOURCES R

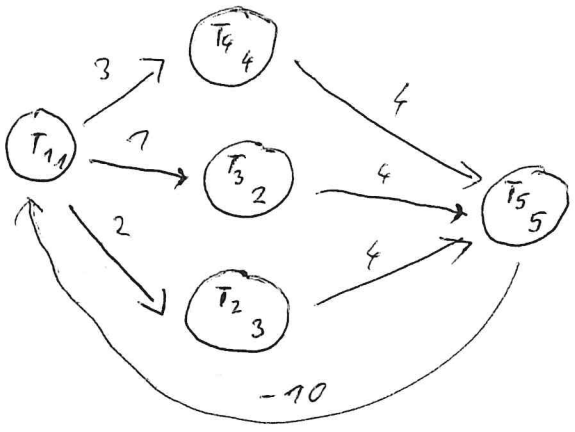
- LPT
  - LONGEST PROCESSING TIME FIRST
  - APPROXIMATION ALGORITHM FOR $P||C_{MAX}$
  - MODIFIED LIST SCHEDULING
    - SORT LIST L BY PROCESSING TIME p_i, LONGEST FIRST

- PROJECT SCHEDULING, MINIMIZING C_{MAX}

- $PS|TEMP|C_{MAX}$
  - NP-HARD
  - INPUT
    - NUMBER n OF NON-PREEMPTIVE TASKS
    - PROCESSING TIMES $(p_1, p_2 \dots p_n)$
    - TEMPORAL CONSTRAINTS DEFINED BY GRAPH G

- $PS_m, 1|TEMP|C_{MAX}$
  - NP-HARD
  - $PS_m, 1$
    - m RESOURCE TYPES
    - WITH CAPACITY 1

- TEMPORAL CONSTRAINTS



- PROCESSING TIME p_i IS ASSIGNED TO EACH TASK
- EDGE REPRESENT TEMPORAL CONSTRAINTS
- TEMPORAL CONSTRAINT IS CHARACTARIZED BY ONE INEQUALITY

$$S_i + l_{ij} \leq S_j$$

- "START OF ONE TASK DEPENDS ON START TIME OF ANOTHER TASK"
- NECESSARY CONDITION FOR SCHEDULABILITY IS ABSENCE OF POSITIVE CYCLE IN GRAPH

- ILP FORMULATION OF $PS1|TEMP|C_{MAX}$

  - TWO REPRESENTATIONS

    - TIME INDEXED

      - BASED ON VARIABLE X_{it}

        - $X_{it} = 1$ IFF $S_i = t$

      - PROCESSING TIMES ARE POSITIVE INTEGERS

    - RELATIVE ORDER

      - X_{ij}

        - $X_{ij} = 1$ IFF T_i PRECEDES TASK T_j

      - PROCESSING TIMES ARE NONNEGATIVE REAL NUMBERS

- BOTH MODELS CONTAIN TWO TYPES OF CONSTRAINTS

  - PRECEDENCE

  - RESOURCE

    - PREVENT OVERLAPPING OF TASKS

- TIME INDEXED

  $\min C_{MAX}$

  $\sum_{t=0}^{UB-1} (t \cdot X_{it}) + l_{ij} \leq \sum_{t=0}^{UB-1} (t \cdot X_{jt}) \quad \forall i,j \; l_{ij} \neq \infty \text{ AND } i \neq j$ (PREC. co

  $\sum_{i=1}^{n} \left(\sum_{s=\max(0, t-p_i+1)}^{t} X_{is} \right) \leq 1 \quad \forall t \in \{0, \dots, UB-1\}$ (RESOURCE)

  $\sum_{t=0}^{UB-1} X_{it} = 1 \quad \forall i \in \{1, \dots n\}$ (T_i IS SCHEDULED)

  $\sum_{t=0}^{UB-1} (t \cdot X_{it}) + p_i \leq C_{MAX} \quad \forall i \in \{1, \dots n\}$

  UB - UPPER BOUND

- RELATIVE-ORDER MODEL

$$\min C_{MAX}$$

$$S_i + l_{ij} \leq S_j \qquad \forall\, l_{ij} \neq -\infty \text{ AND } i \neq j \quad \text{(TEMPORAL CONSTRAINT)}$$

$$P_j \leq S_i - S_j + UB \cdot X_{ij} \leq UB - P_i \qquad \forall i,j \in \{1,\dots,n\} \text{ AND } i < j \quad \text{(RESOURCE CONSTRAINT)}$$

$$S_i + P_i \leq C_{MAX} \qquad \forall\, i \in \{1,\dots,n\}$$