

## General Purpose Methods for Combinatorial Optimization

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

1

### Maximum Contiguous Sum

31	-41	59	26	-53	58	97	-93	-23	84
----	-----	----	----	-----	----	----	-----	-----	----

$$\Sigma = 187$$

Given:  $A_1 \dots A_N \in \mathbb{Z}$ , at least one  $A_i > 0$

Find  $i, j$  such that  $\sum_{k=i}^j A_k$  is maximal

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

2

## MCS: Algorithm 2

31	-41	59	26	-53	58	97	-93	-23	84
----	-----	----	----	-----	----	----	-----	-----	----

↑
↑  
L
U

```

MaxSoFar := 0
for L := 1 to N do
  sum := 0
  for U := L to N do
    sum := sum + x[U]
    ← sum of x[ L ... U ]
  MaxSoFar := max (MaxSoFar, sum)

```

Faster: one nested for-loop less

## Analysis of Algorithm 2

```

MaxSoFar := 0
for L := 1 to N do
  sum := 0
  for U := L to N do
    sum := sum + x[U]
    ← sum of x[ L ... U ]
  MaxSoFar := max (MaxSoFar, sum)

```

- Constant time per inner loop iteration
- Count number of times inner loop is executed

- $\sum_{L=1}^N \sum_{U=L}^N O(1) = O(N^2)$

## Challenge: Algorithm 3

- Can you develop a linear time algorithm?
- Solution ~~next~~ lecture

this

31	-41	59	26	-53	58	97	-93	-23	84
----	-----	----	----	-----	----	----	-----	-----	----

$$\Sigma = 187$$

## Algorithm 3: Linear Time (solution)

31	-41	59	26	-53	58	97	-93	-23	84	
31	0	59	85	32	90	187	94	71	155	MaxEndingHere
31	31	59	85	85	90	187	187	187	187	MaxSoFar

- Empty vector gives maximum sum == 0
- Scan the array just once
- Keep track of max sum in first I elements (-> MaxSoFar)
- And of maximum sum ending in position I (-> MaxEndingHere)

```

MaxSoFar := 0
MaxEndingHere := 0
for I := 1 to N do
  MaxEndingHere := max (0, MaxEndingHere + x[I])
  MaxSoFar := max (MaxSoFar, MaxEndingHere)

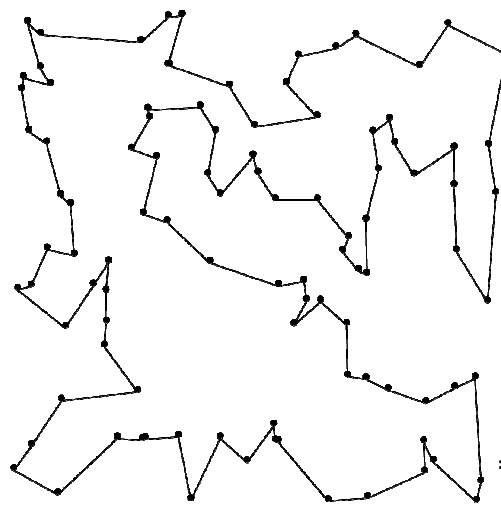
```

## Comparison

algorithm	1	2	3
computation time ( $\mu\text{sec}$ )	$3.4 N^3$	$13 N^2$	$33 N$
computation time as a function of N	$10^2$	$10^3$	$10^4$
	3.4 sec	130 ms	3.3 ms
	$10^3$	1 hour	13 sec
	$10^4$	39 days	22 min
	$10^5$	108 years	1.5 days
	$10^6$	1080 centuries	5 months
			33 sec

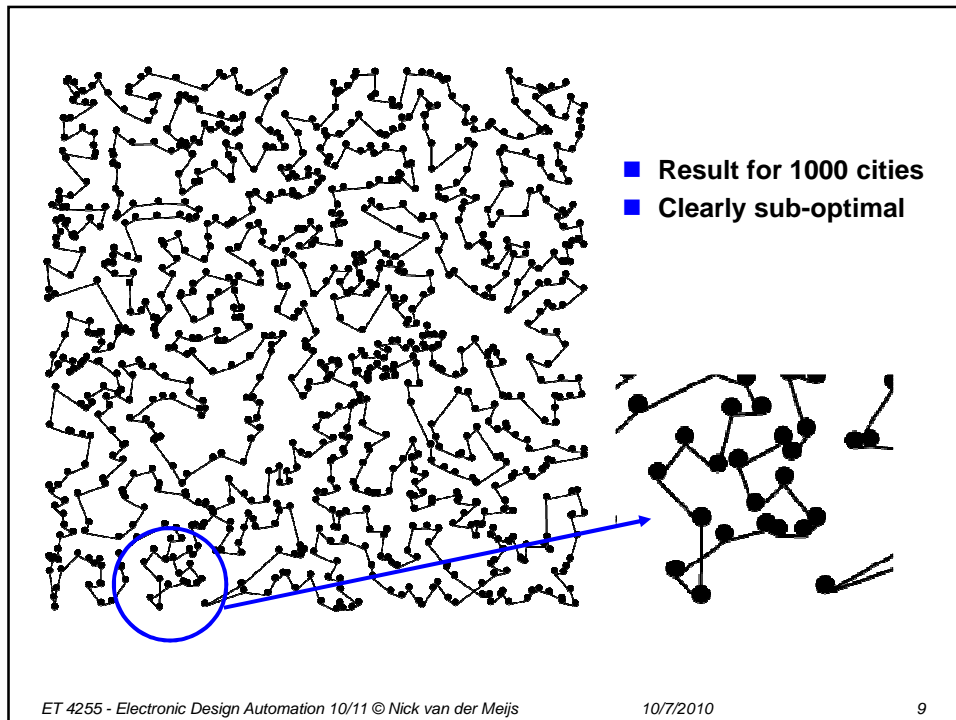
## Traveling Salesman Problem

- No polynomial algorithm known
- Exact solution only by exhaustive search
- $\frac{1}{2}((N-1)!)$  possible routes



Stirling:  
 $\ln n! \approx n \log n - n$

Large n:  
 $n! \approx e^{n \log n}$   
 $= (e^{\log n})^n = n^n = o(e^n)$



## Simulated Annealing

- important
- much applied method for various combinatorial optimization problems.

Idea (thermo dynamics, static mechanics)

Growing of crystals by means of Annealing

1. start off with high temperature
2. cool down slowly

Theoretical Model

- matter strives for state with lowest energy
  - movement of atoms  $\Leftrightarrow$  small random displacements
- $\Delta E < 0 \Rightarrow$  accept displacement
- $\Delta E > 0 \Rightarrow$  accept with probability  $\exp(-\Delta E/k_B T)$



## Simulated Annealing (2)

state of atoms  $\leftrightarrow$  configuration  
 displacement  $\leftrightarrow$  move  
 energy  $\leftrightarrow$  cost function  
 perfect crystal  $\leftrightarrow$  optimal solution

### Algorithm

```

T := T0
X := start configuration
while (not satisfied stop criterium) {
  for (a few times) {
    Xnew := new configuration
    if (accept (cost (Xnew) - cost (X), T))
      X := Xnew
  }
  T := update (T)
}

accept ( $\Delta_{\text{cost}}, T$ ) {
  if ( $\Delta_{\text{cost}} < 0$ )
    return TRUE
  else {
    if ( $\exp(-\Delta_{\text{cost}}/cT) > \text{random}(0,1)$ )
      return TRUE
    else
      return FALSE
  }
}
  
```

## Traveling Salesman by means of Simulated Annealing

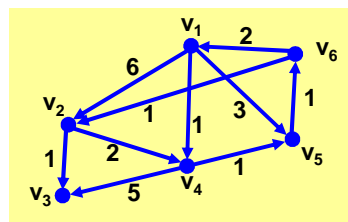
- Randomly select two cities, interchange order of visiting:  
A - B - C - D  $\rightarrow$  A - C - B - D
- relatively long calculations
- be careful with detailed parameters of algorithm

## Shortest Path

Given: Weighted, directed graph  $G=(V,E)$

1. Single-source shortest path  
Find shortest path from source vertex  $s \in V$  to each  $v \in V$
2. Single-destination shortest path  
equivalent by reducing (reverse direction of edges)
3. Single-pair shortest path  
route-planning  
no known algorithm being asymptotically better than 1
4. All-pairs shortest path

## Dijkstra's Shortest Path Algorithm



**Q:** What is the shortest path from  $v_1$  to  $v_2$ ?

**A:**  $v_1 - v_4 - v_5 - v_6 - v_2$   
of length 4

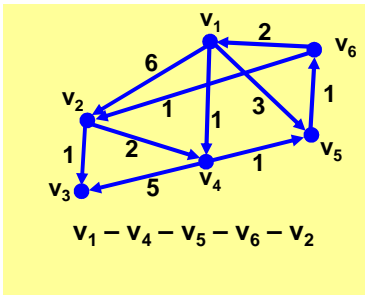
1. Find from  $v_1$  the shortest path to its neighbors
2. Say  $u$  is the closest to  $v_1$
3. See if any of the routes from  $v_1$  to the neighbors of  $u$  becomes shorter if passing through  $u$
4. Continue with step 2, until reaching  $v_2$  (target)

## Dijkstra's Shortest Path Algorithm

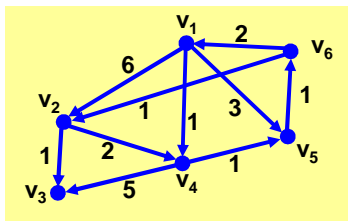
dijkstra(set of struct vertex  $V$ , struct vertex  $v_s$ , struct vertex  $v_t$ )

```

{
  set of struct vertex  $T$ ;
  struct vertex  $u, v$ ;
   $V \leftarrow V \setminus \{v_s\}$ ;
   $T \leftarrow \{v_s\}$ ;
   $v_s.distance \leftarrow 0$ ;
  for each  $u \in V$ 
    if  $((v_s, u) \in E)$ 
       $u.distance \leftarrow w((v_s, u))$ 
    else  $u.distance \leftarrow +\infty$ ;
  while  $(v_t \notin T)$  {
     $u \leftarrow$  "u  $\in V$ , such that  $\forall v \in V : u.distance \leq v.distance$ ";
     $T \leftarrow T \cup \{u\}$ ;
     $V \leftarrow V \setminus \{u\}$ ;
    for each  $v$  "such that  $(u, v) \in E$ "
      if  $(v.distance > w((u, v)) + u.distance)$ 
         $v.distance \leftarrow w((u, v)) + u.distance$ ;
  }
}
    
```



## Dijkstra's Shortest Path Algorithm



Shortest distance to these vertices is known

1. Find from  $v_1$  the shortest path to its neighbors
2. Say  $u$  is the closest to  $v_1$
3. See if any of the routes from  $v_1$  to the neighbors of  $u$  becomes shorter if passing through  $u$
4. Continue with step 2, until reaching  $v_2$  (target)

iteration	$T$	$v_i.distance$ , for $i =$					
		1	2	3	4	5	6
1	$\{v_1\}$	6	$\infty$	<u>1</u>	3	$\infty$	
2	$\{v_1, v_4\}$	6	6		<u>2</u>	$\infty$	
3	$\{v_1, v_4, v_5\}$	6	6			<u>3</u>	
4	$\{v_1, v_4, v_5, v_6\}$	<u>4</u>	6				
5	$\{v_1, v_4, v_5, v_6, v_2\}$		<u>5</u>				
6	$\{v_1, v_4, v_5, v_6, v_2, v_3\}$						

This is vertex  $u$ . Shortest distance to these vertices has just become known

### Dijkstra in PYTHON language

```

def Dijkstra (G, start, end = None, animate=0):
    # G[v] is a dict of distances keyed by vertices adjacent to v
    # G[v][u] is the distance from v to u if u is adjacent to v
    T = set ()                # set of vertices for which min distance is known.
    D = dict ()              # dictionary of final distances,
                            # with vertex as key and distance as value

    V = G.keys ()           # set which is initialized to all vertices.

    T.add (start)
    V.remove (start)

    D[start] = 0            # initialize all distances
    for v in V: D[v] = sys.maxint

    for v in G[start]: D [v] = G[start][v] # loop over all vertices reachable from start

    while end not in T and len (V) > 0:
        if animate: print "T", T, "\n", "D", D
        u = closest (D, V) # find closest vertex to start, among those in V
        T.add (u)
        V.remove (u)
        for v in G[u]:
            if D [v] > G[u][v] + D[u]:
                D [v] = G[u][v] + D[u]

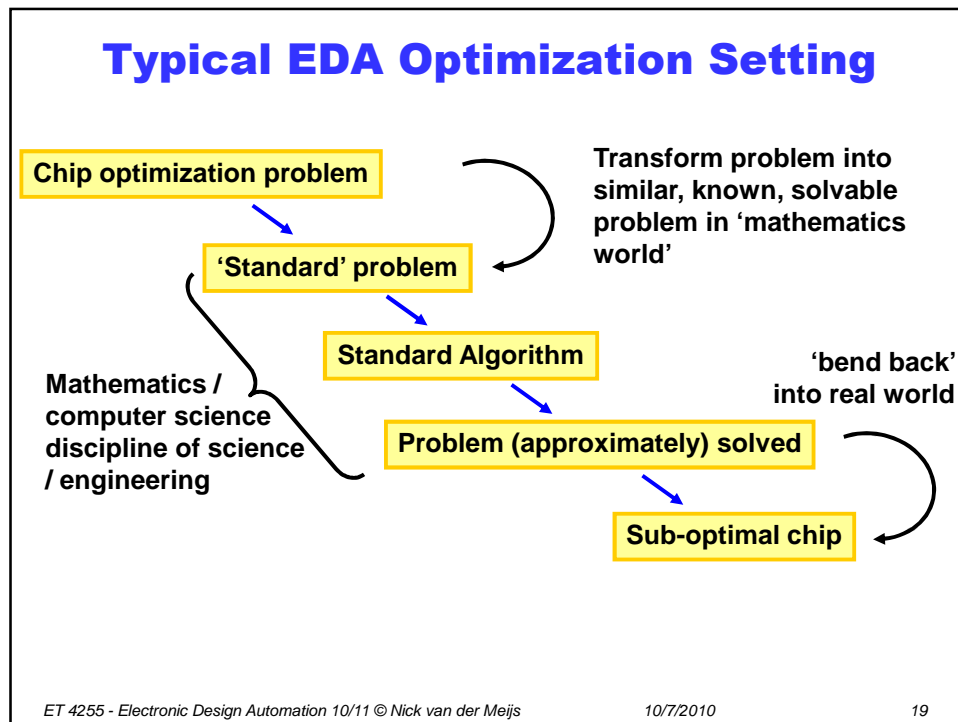
    return D

```

[www.python.org](http://www.python.org)

## Optimization

- **Public Transport:**
  - It brings you from a place where you are not, to a place where you don't want to be, at a time that you do not prefer
- **EDA:**
  - It provides a solution to a problem that you don't want to solve
  - Why?
    - Because most EDA problems are just too hard to be solved exactly
    - Many optimization problems are NP-hard
    - Therefore, problems are usually replaced by simpler problems that are solved instead.

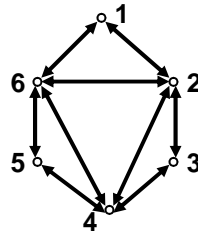


## Continuous vs Discrete Optimization

- **Continuous Optimization:** Solution space forms a continuous domain
  - Example: adjust transistor sizes in a network to optimize the speed of the network, under the assumption of continuously variable transistor dimensions
- **Discrete Optimization:** the number of solutions can be counted, from a discrete set (might be infinite)
  - Example: traveling salesman in graph – there are a finite number of distinct tours visiting all vertices
  - Or: adjust transistor sizes in a network to optimize the speed of the network, under the assumption that transistor dimensions should be integral multiples of a 'step size'.

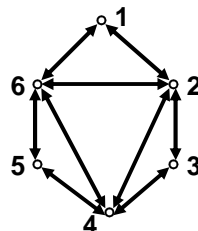
ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 20

## Discrete Optimization



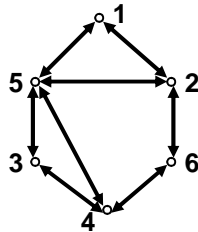
- Optimization algorithms typically go through a sequence of steps
- Set of choices at each step
- (Result of) each step is called a configuration
- Above we see 6 configurations with different cost, 4 choices in config of cost 6.

## Discrete Optimization (2)



- |                     |   |
|---------------------|---|
| $\Pi$               | given optimization problem  |
| $p$                 | instance of $\Pi$   |
| $A$                 | algorithm for $\Pi$   |
| $X_p = X = \{x_k\}$ | set of all possible (legal) configurations (solutions) of $p$         |
| $c(x_k)$            | cost of configuration $x_k$   |
| $G = (X, E)$        | a directed graph, the <b>configuration graph</b>                      |
| step (move)         | the transformation (by the algorithm) of configuration $x_i$ in $x_j$ |
| $E$                 | set of all steps that algorithm $A$ considers                         |

## Discrete Optimization (3)



- What is the global minimum?
- Is this example convex?
- If not:
  - What is/are local minima?
  - Can you change the graph to become convex?

**local minimum**

$x_i$  is a local minimum in case no neighboring configurations exist with lower cost

**global minimum**

desired configuration with absolute lowest cost

**greedy algorithm**

algorithm that only takes steps that result in a lower cost

**convex configuration graph**

in case every local minimum also global minimum (and strongly connected)  $\Rightarrow$  greedy algorithm will give exact solution. (Not necessarily in optimal time.)

**Uphill move**

step of algorithm that (temporarily) increases cost aimed at escaping local minima

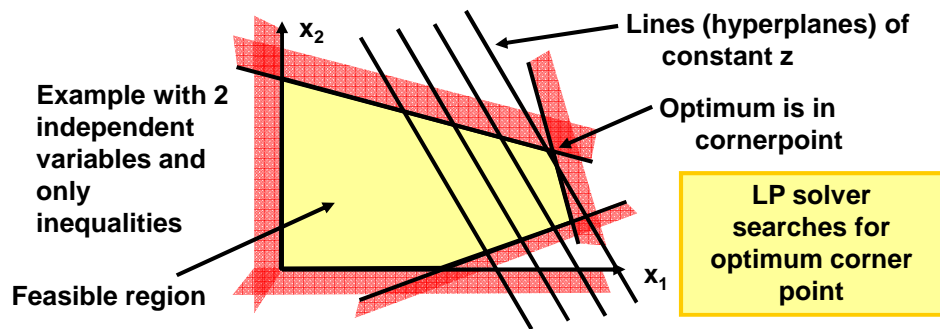
## Standard Optimization Techniques

- Backtracking, branch and bound ( $\rightarrow$  § 5.2)
- Dynamic programming ( $\rightarrow$  § 5.3)
- Linear programming (LP), integer LP (ILP) ( $\rightarrow$  § 5.4)
- Local Search ( $\rightarrow$  § 5.5)
- Simulated Annealing ( $\rightarrow$  § 5.6)
- Tabu search ( $\rightarrow$  § 5.7)
- Genetic algorithms ( $\rightarrow$  § 5.8)
- Boolean Satisfiability (SAT)
- Neural networks
- Simulated evolution
- Matching, max flow, shortest path,
- Non-linear programming: Lagrange relaxation, Levenberg-Marquardt, ...
- ...



## Linear Programming (LP)

- Maximize  $z = x_1 + x_2 + 3x_3 - \frac{1}{2}x_4$  (objective function)
  - Such that  $x_i > 0$  for all  $i$  (primary constraints)
  - $x_1 + 2x_3 \leq 740$
  - $2x_2 - 7x_4 \leq 0$
  - $x_2 - x_3 + 2x_4 \geq \frac{1}{2}$
  - $x_1 + x_2 + x_3 + x_4 = 9$
- } (additional constraints)



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

27

## LP Solvers

- Ellipsoid algorithm runs in polynomial time
- In practice, Simplex algorithm is often faster, but it has exponential worst-case time complexity
- Simplex algorithm is clever way of enumerating boundary points, it steps from point to point along the outside of the feasible region
- Based on principles from linear algebra
- Many on-line sources (e.g. see wikipedia -> linear programming)
- Also see the book 'Numerical Recipes in {C/C++/Fortran}' – Press et al. On-line C version of the book (pdf) via [www.nr.com](http://www.nr.com)
- See [lp\\_solve](#) originally by Michel Berkelaar ⇒ Now at TUD/EWI/CAS

NUMERICAL RECIPES  
Books On-Line

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

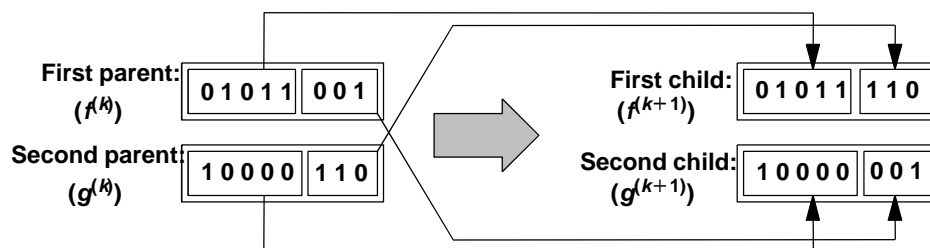
10/7/2010

28

## Integer Linear Programming (ILP)

- Linear Programming with the  $x_i$  restricted to **integer numbers**
- Surprise: **ILP is NP-hard** (LP has polynomial time complexity)
- **Does not work**: solving an 'equivalent' LP and rounding to nearest integer; solution may not be optimal, or even feasible.
- Yet other variant is **0-1 LP**,  $x_i \in \{0, 1\}$

## Genetic Algorithms



- Inspired on biology: **survival of the fittest**
- Work with a **population**  $P^{(k)} \subset F$ , **set of feasible solutions**
- Each  $f \in P^{(k)}$  is encoded as a chromosome, e.g. a bit-string
- **Cross-over operations** between members of  $P^{(k)}$  to derive  $P^{(k+1)}$
- Prefer parents with lower cost for 'mating'
- Many variations possible (mutation, cloning, ...)

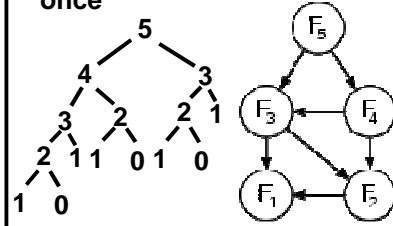
## Dynamic Programming

- Fibonacci numbers:  
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

```
function fib(n)
  if n = 0 or n = 1 return n
  else return fib(n - 1) + fib(n - 2)
```

```
var f := map(0 → 0, 1 → 1)
function fib(n)
  if n not in keys(f)
    f[n] := fib(n - 1) + fib(n - 2)
  return f[n]
```

Recursive algorithm is a lot of work: e.g. fib(2), fib(3) computed more than once



Dynamic programming:

- Overlapping subproblems
- Optimal substructure
- Memoization

## Fibonacci Sequence Using Dynamic Programming in Python

```
f = {0: 0, 1: 1}
def dynfib (n):
  if n not in f:
    f[n] = dynfib (n-1) + dynfib (n-2)
  return f[n]
```

- Recursive version
- Time and space  $O(n)$
- Runs into max recursion depth problems

```
def dynfib2 (n):
  if n == 0 or n == 1: return n
  prevprev = 0
  prev = 1
  for i in range (2, n+1):
    fib = prev + prevprev
    prevprev = prev
    prev = fib
  return fib
```

- Iterative (bottom-up) version
- Time  $O(n)$ , space  $O(1)$
- Runs easily for  $n = 100000$

## Dynamic Programming

- Elegant approach
- Two variants: bottom-up and top-down
- Several applications in EDA
- E.g. in channel routing, later lecture.
- E.g. technology mapping, Dirk-Jan Jongeneel,

Optimized boolean equations

Library of std cells

Timing constraints



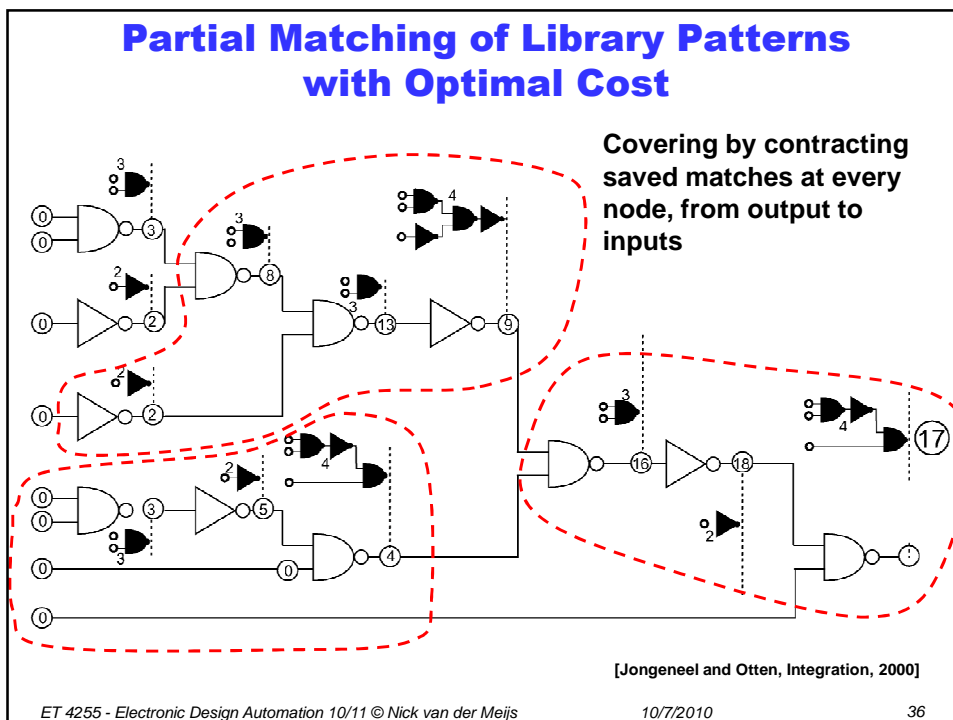
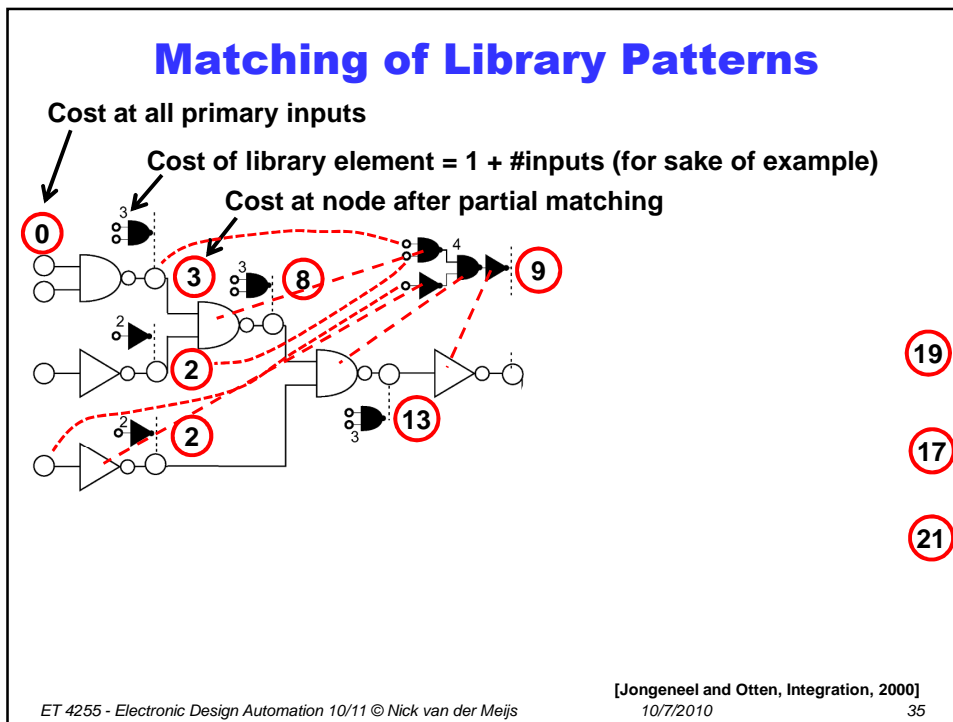
Gate netlist

[Jongeneel and Otten, Integration, 2000]

## Library of Std Cells

library function								
cost parameters	$A, \alpha, \beta, \gamma$	$A, \alpha, \beta, \gamma$	$A, \alpha, \beta, \gamma$	$A, \alpha, \beta, \gamma$	$A, \alpha, \beta, \gamma$	$A, \alpha, \beta, \gamma$	$A, \alpha, \beta, \gamma$	$A, \alpha, \beta, \gamma$
Patterns								

[Jongeneel and Otten, Integration, 2000]



### Techn. Map. Result

[Jongeneel and Otten, Integration, 2000]

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 37

### Summary

- Graph algorithms and optimization very common in EDA
- Discrete vs continuous optimization

- Many different algorithms, with different properties
- Finding “right” one depends on transformation from and back into “real world”

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 38

## Summary

### Many different optimization algorithmic techniques

- Backtracking, branch and bound (→ 5.2)
  - Dynamic programming (→ 5.3)
  - Linear programming (LP), integer LP (ILP) (→ 5.4)
  - Local Search (→ 5.5)
  - Simulated Annealing (→ 5.6)
  - Tabu search (→ 5.7)
  - Genetic algorithms (→ 5.8)
  - Boolean Satisfiability (SAT)
  - Neural networks
  - Simulated evolution
  - Matching, max flow, shortest path,
  - Non-linear programming: Lagrange relaxation, Levenberg-Marquardt, ...
  - ...
- 
- Many applications of optimization in EDA (e.g. techn. mapping)
  - Asymptotic complexity important for effectivity of algorithm
  - Implementation can also count (e.g. recursive vs iterative implementation)