CEG5270 Assignment 1 (2009 Spring)

XIAO Zigang
zgxiao@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong

February 16, 2009

1. Use greedy algorithm to solve this problem.

Algorithm 1 Find chromatic number of interval graphs

```
1: Sort the start and end point of intervals, save it in a list $L$.
2: Mark all intervals as not colored.
3: freelist $\leftarrow \{1, 2, ..., n\}$, usedlist $\leftarrow \varnothing$, $k \leftarrow 0$
4: for $i = 1$ to $2n$ do
5:     if $L_i$ is start point of interval $x$ then
6:         $c \leftarrow$ head of freelist
7:         $x$.color $\leftarrow c$
8:         freelist $\leftarrow$ freelist $-\{c\}$
9:         usedlist $\leftarrow \{c\}\cup$ usedlist
10:        if $c > k$ then
11:            $k \leftarrow c$
12:        end if
13:    else // $L_i$ is end point of interval $x$
14:        $c \leftarrow x$.color
15:        usedlist $\leftarrow$ usedlist $-\{c\}$
16:        freelist $\leftarrow \{c\}\cup$ freelist
17:    end if
18: end for
19: return $k$.
```

Proof: Let x be the vertex which is colored to k. Since x is not colored to a smaller value, it means x's start point a intersects with the intervals which are colored from 1 to $k-1$, i.e. those intervals includes a. Then these intervals forms a $k-$clique. Hence $\omega(G) \geq k \geq \chi(G)$.But we know that $\chi(G) \geq \omega(G)$. Hence $\omega(G) = \chi(G) = k$.

Complexity: Sorting $2n$ elements takes $O(n \log n)$. And it takes $O(n)$ for the for loop in the algorithm, only $O(1)$ for the inserting and removing operation of queues. Hence the total complexity is dominated by the sorting time $O(n \log n)$.

2. Let $H[k][i][j]$,$V[k][i][j]$ denote the number of at most K multi-bend routes from s to t where the last step is horizontal move and vertical move, respectively. And:

$$H[k][i][j] = \begin{cases} 0 & \text{if } j = 0 \text{ or } i \neq 0 \wedge j \neq 0 \wedge k = 0 \\ 1 & \text{if } j \neq 0 \wedge i = 0 \\ H[k][i][j-1] + V[k-1][i][j-1] & \text{otherwise} \end{cases} \tag{1}$$

$$V[k][i][j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } i \neq 0 \wedge j \neq 0 \wedge k = 0 \\ 1 & \text{if } i \neq 0 \wedge j = 0 \\ H[k-1][i-1][j] + V[k][i-1][j] & \text{otherwise} \end{cases} \tag{2}$$

Without loss of generality, let $(0,0)$ and (x,y) be the coordinate of s,t, then $H[k][x][y] + V[k][x][y]$ is the number of multi-bend routes from s to t.

3. (i)

Algorithm 2 Computer coordinate of PE modules

```
1: Define a tree structure where each subtree represent a block
2: Initialize each operand and operator as a tree node
3: for $i = 1$ to $2n-1$ do //construct the tree from the normalized expression $\alpha$
4:    if $\alpha_i$ is an operand then
5:       stack.push($\alpha_i$)
6:    else //$\alpha_i$ is an operator
7:       b ← stack.pop(), a ← stack.pop()
8:       construct a new node where $\alpha_i$ is the parent and a,b is the children
9:    end if
10: end for
11: head ← stack.pop()
12: BFS(head) //use a BFS to compute the corner of each block.
```

Algorithm 3 Function BFS(head)

```
1: queue.enque(head)
2: while queue not empty do
3:    node ← queue.pop_head()
4:    L ← node.left_child, R ← node.right_child
5:    if node == '*' then
6:       L.(x,y) ← (node.x, node.y)
7:       R.(x,y) ← (node.x + L.width, node.y)
8:       queue.enque(L), queue.enque(R)
9:    else if node == '+' then
10:      L.(x,y) ← (node.x, node.y)
11:      R.(x,y) ← (node.x, node.y + L.height)
12:      queue.enque(L), queue.enque(R)
13:   end if
14: end while
```

Analysis:

In the first step, we traverse the polish expression to construct the tree, which takes $O(n)$ time. After the tree is constructed, the BFS also takes $O(n)$ time because there are $n-1$ operators(internal nodes) in the tree. Hence the total run time of the algorithm is $O(n)$.

(ii) Proof:

The first move ensures that given n operands, we can switch any two operands by one or more such moves, i.e. we can get all permutations of operands;

The third move ensures that operator can switch to any 'slot' between two operands(may not be valid polish expression). Also note that we could always generate a polish expression in the form $p_1p_2e_1p_3e_2...p_ne_{n-1}$, i.e. all operator forms a chain of size 1 by itself. Then using the second move, we can generate all possible combination$(=2^{n-1})$ of operators. By combining the third second move, we can generate all possible permutation of operators.

Since every valid normalized polish expression is corresponds to a slicing floorplan, which means the solution space is a subset of the possible string generated by the three moves, hence the reachability is ensured.

4. (i) $C : q[j] < i$ and no edge between $q[j]$ and $Largest$
$D : Largest = q[j]$

Explanation: Condition C means that if for some number $x = q[j] < i$, then they have the relationship like $(...x...i...)$, $(...x...i...)$ in the sequence pair. Also it checks if it will be a transitive edge. Statement D helps to update the $Largest$ so as to keep track of the transitive edge information.

(ii) Change line 3 to:
3: For(j=k+1 up to n) do

Replace every occurrence of G_h to G_v in the pseudo code.

(iii) Since every pair of modules has a binary relation, the total number of edges would be C_n^2 with transitive edges. In the worst case of horizontal constraint graph, all the modules are placed one next to another. But if transitive edges are removed there will be at most $(n-1)$ edges. This result holds in vertical constraint graph. Hence the number of edges in the constraint graph can be reduce to $O(n)$.

5. (i) For 1: (4,12), (6,8), (8,6), (12,4)

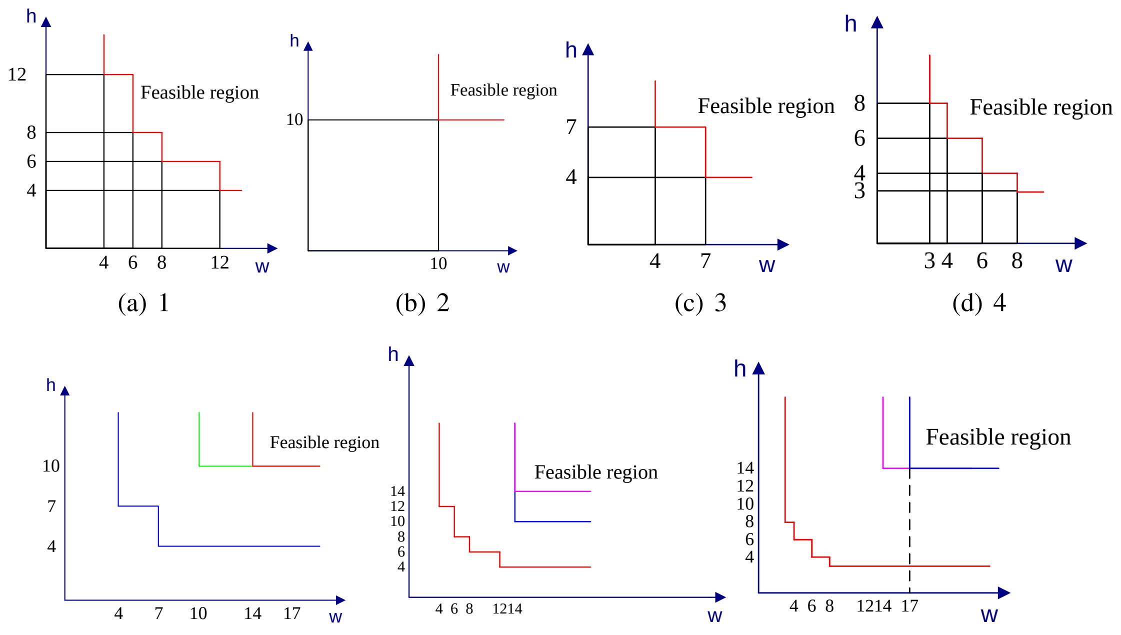
For 2: (10,10)

For 3: (4,7), (7,4)

For 4: (3,8), (4,6), (6,4), (8,3)

(ii) See Figure 1.

(iii) According to the shape curve of $123*+4*$, the width and height of the smallest packing should be 17×14 .



(a) 1 (b) 2 (c) 3 (d) 4

(e) 23* (f) 123* (g) 123*+4*

Figure 1: shape curves