Last Section

分支定界

- 主要的分支定界法相关概念:

  Feasible （solution）, Infeasible （solution）, Partial Solution, Optimal Solution, Branch, Bound (tight), Traverse, Backtracking, Prune, DFS, BFS, Frontier Search, Initial solution, Relax

- 一般着色问题

- 分支定界法解整数规划问题

- 背包问题

分支定界

- TSP
- 分配（指派）问题
- 匈牙利法
- 一些说明

Branch and Bound

Minimum Cost Network Flow Problem

问题描述

- 该类设计问题的目标是，在一组节点之间设计一最小耗费的网络以满足所有的流量需求。

- 描述该问题的信息：
  - 每个源节点和目的节点对(O-D)之间的流量需求
  - 在每条边(i,j)上负载流量的价值函数

    $C_{ij}=f(t_{ij})$

问题描述

- 设**G**表示在n个点上的所有无向图的集合，G是该集合中的一个元素。

- 该问题就是要找到一个$G^* \in \mathbf{G}$，使得在满足给定的边容量限制的前提下，实现传输所有源节点到目的节点的流量需求所用的花费最小。

The Problem

- Formulation-1
- Interpretation of the formulation
- Too complicated? Then what?

Transform and conquer

- Problem transformation
  - Constant arc costs
  - Tree solution

Consider a connected undirected graph $G = (V, A, d, c)$ with node set $V = \{1,2,...,n\}$ and arc set A.

There is a certain amount of traffic demand d_{ij} between each pair of nodes i and j in V.

c_{ij} represents the cost of using arc (i, j) in A.

To satisfy the capacity constraint, the flow on any arc (i, j) must not exceed a given capacity k_{ij}.

By means of these definitions, the problem is to find a minimum cost tree spanning the node set V where all traffic demand are satisfied, and the capacity constraint k_{ij} is observed for every arc (i, j).

Let f_{ij} denote the total flow on arc (i, j). Define $x_{ij} = 1$, if arc (i, j) is included in the solution, and $x_{ij} = 0$, otherwise.

给定一个连通的无向图G=(V,A,d,c,k)，

点集V={1,2,…,n}，边集A.

V中的每个节点对i和j间，有一定的流量需求d_{ij}.

c_{ij}表示建立A中的边(i,j)的花费。

每条边(i,j)上的流量负载不能超过一个给定值k_{ij}.

该问题就是要找到一个可以扩展节点集V的生成树，使得在满足所有的流量需求，并且每条边(i,j)都符合负载流量限制k_{ij}的前提下，花费最小。

令f_{ij}表示经过边(i,j) 的总流量。定义

x_{ij}=1,当边(i,j)被选中在解中；

x_{ij}=0,当边(i,j)没有被选中。

Problem II

- Formulation-2

- Interpretation of the formulation

Branch and Bound

- What is the first task?

Branch and Bound

- What is the first task?
- Constructing a search tree
- How to construct the search tree?

Branch and Bound

- What is the first task?
- Constructing a search tree
- How to construct the search tree?

  -- Innumerate，Organize，Bound

Branch and Bound

The search tree

- a tree spanning n nodes must have exactly n-1 arcs
- consider the solution as a collection of n-1 arcs (for n-node problem) chosen from all candidate arcs
- introduce an arc-oriented branch and bound method, branching by using an arc or not.

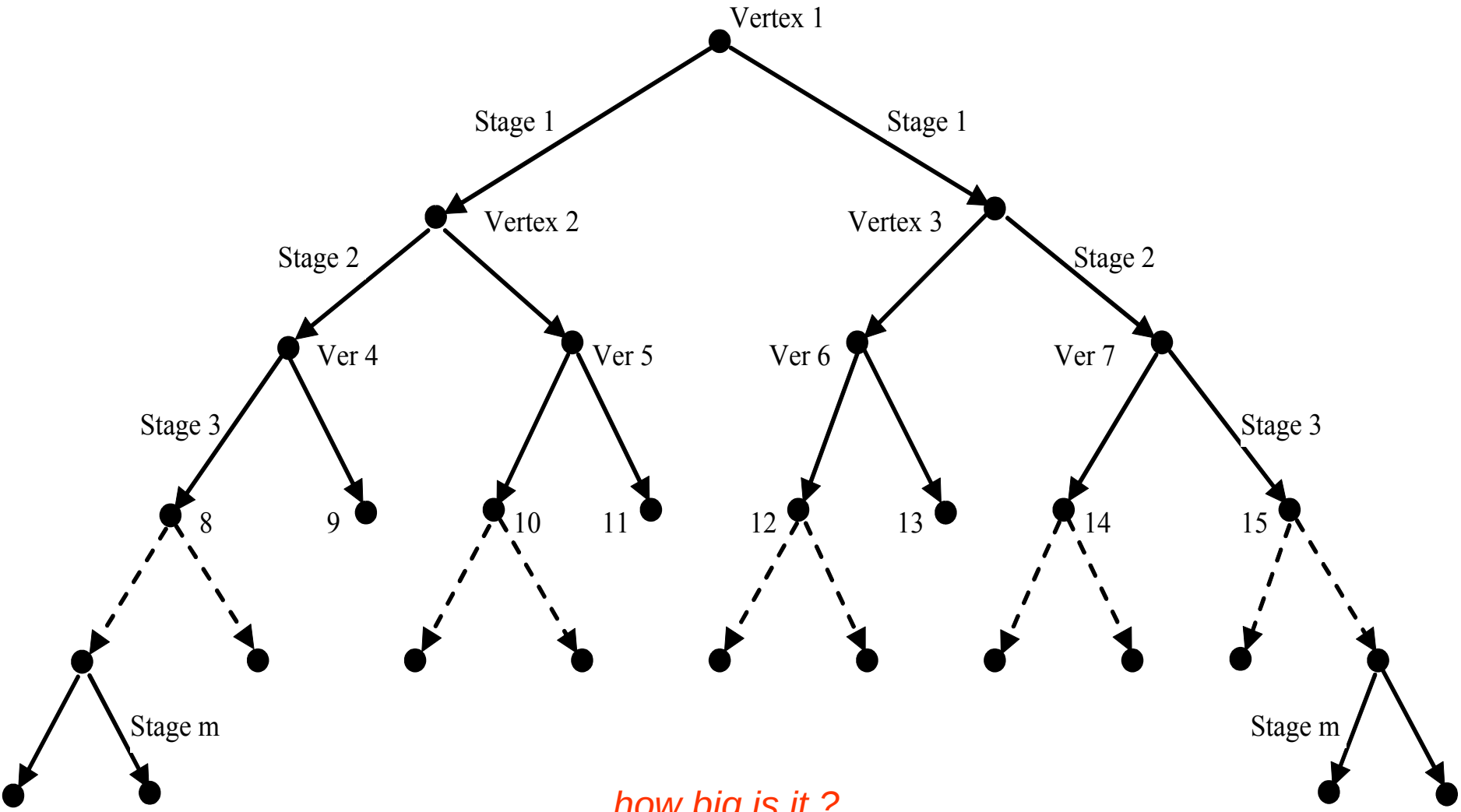
A Binary Search Tree

- Define an m-stage binary search tree for problem with n nodes, where, for complete graph problem, m equals the number of pairs of nodes.
- Give each of the arcs an arc number, from 1 to m, as identification.
- The branching within each stage corresponds to the decision of either adding a certain arc to the solution arc set or not.
- Define traveling along left branch represents for choosing an arc and traveling along right branch represents for not choosing an arc.

二叉搜索树

- 对于n个节点的问题定义一个m层的二叉搜索树，对于完全图问题，m等于节点对的个数。
- 为每条边编号，从1到m。
- 每层的分支对应于是否把某条边添加到解集中的决策。
- 定义沿左分支前进表示选择该边，沿右分支前进表示不选该边。

The search tree



how big is it ?

The search tree

- KEEP The Tree in Mind

The search tree

- KEEP The Tree in Mind
- What’s next?

The search tree

- KEEP The Tree in Mind
- What's next?
- Regulations for traversing the search tree.
  - DFS
  - BFS
  - FS

Traversing the search tree

It is obvious that an optimal solution can be found after having visited all the vertices of this binary tree.

1. Always start from the root of the tree (Vertex 1).
2. Always go along the left branch first when descending.
3. Keep descending and backtrack when any of the following situations occurs:
   a. Current selected arcs make the solution infeasible.
   b. A feasible solution has been found.
4. After backtracking to a vertex along its left branch, go forward along its right branch.
5. After backtracking to a vertex along its right branch, go backward to its parent vertex.

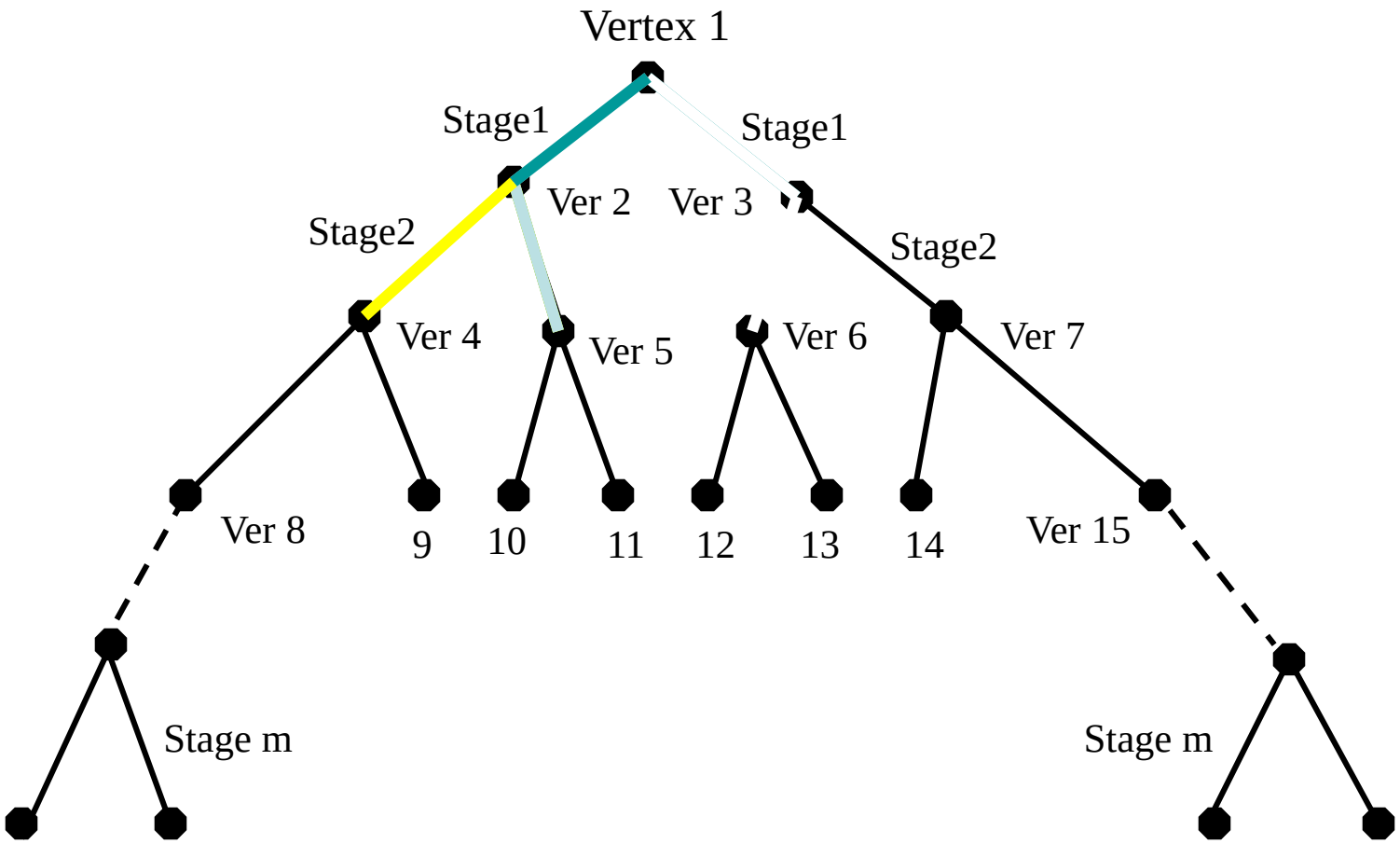
遍历搜索树

明显地，当该二叉搜索树中的所有顶点都被访问过时，可以找到一个最优解。

1. 总是从树的根(Vertex 1)开始。
2. 当下行时，总是先沿左分支进行。
3. 当有如下情况之一发生时，进行回溯：
   a. 当前挑选的边使得解不可行。
   b. 已经找到一个解。

4. 当从左分支回溯到某顶点时，接着沿其右分支向下进行。
5. 当从右分支回溯到某顶点时，接着回溯到其父顶点。

Traversing the search tree



Traversing the search tree

- What’s next?

Traversing the search tree

- What's next?

- Pruning the search tree

Traversing the search tree

- What's next?
- Pruning the search tree
- How?

Traversing the search tree

- What's next?
- Pruning the search tree
- How?
- **Bound**

Upper Bound

Upper Bound

Total cost of an existing solution.

目前已经得到的一个解的总花费

Updated whenever a better solution is found

Lower Bound

Lower Bound

- How ?

Lower Bound

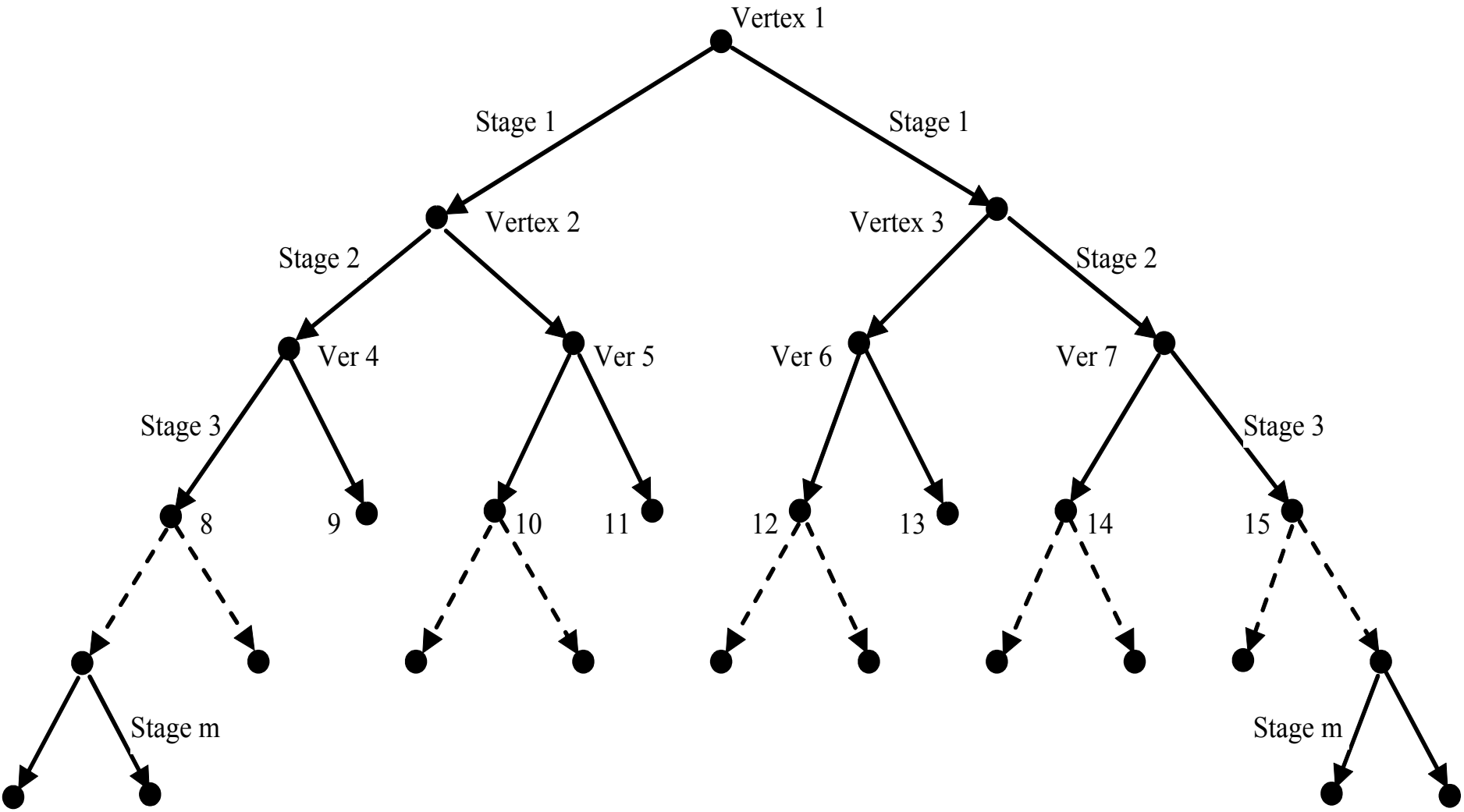
- How ?

- Relax !

Lower Bound

- How ?
- Relax !
- 该问题就是要找到一个可以扩展节点集V的生成树，使得在满足所有的流量需求，并且每条边(i,j)都符合负载流量限制k_{ij}的前提下，花费最小。

The search tree



下界

- 搜索树的每个节点代表由原始问题得到的一个子问题。
- 子问题和整个问题的区别在于，在子问题中，有些边已被定义为不选，而另外一些边则必须选用。
- 如果取消流量限制，那么子问题很类似于传统的最小生成树(MST)问题。
- 这里，我们称取消流量限制的子问题为SMST问题。

A Modified Kruskal's Algorithm for the SMST problem

The Modified Kruskal's Algorithm starts from an empty solution arc set.

Firstly, arcs that are decided not to be used are eliminated from the whole candidate arc set.

Secondly, arcs that must be used are included in the solution arc set; these arcs will not make cycles to the solution tree, since at each vertex in the search tree, it is always checked whether a cycle is made after the correspondent arc is added to the solution.

Next, the cheapest arc in the candidate set is chosen, and the feasibility after it is included in the solution is checked. If the arc forms a cycle to the solution tree, it is abandoned; if it does not form any cycle, it is included in the solution arc set permanently.

Then the second cheapest arc is considered, and this procedure is repeated until a complete tree spanning all nodes is constructed.

SMST问题的Modified Kruskal's算法

从空的解集开始：

首先，把那些被定义为不选的边从整个候选边集合中移除；

然后，把那些必选的边包含到解中（这些边不会在解树中上产生回路，因为在搜索树的每个顶点，当添加相应边时，都会做回路检查。）；

接下来，选择候选边集合中权值最小的边，并检查如果把该边添加到解集中是否满足树条件。如果它在解空间树上形成回路，那么放弃该边；否则，把这条边添加到解集中。

接着考察下一个权值最小的边；

重复上述步骤直到一个V集上的生成树已经构成。

Modified Kruskal's Algorithm

- ***Theorem 1*** The solution found by the Modified Kruskal's Algorithm is the optimal solution for the SMST problem.

- ***Proof*** To prove the optimality of this solution, the method of "reduction to absurdity" is used.

- proof

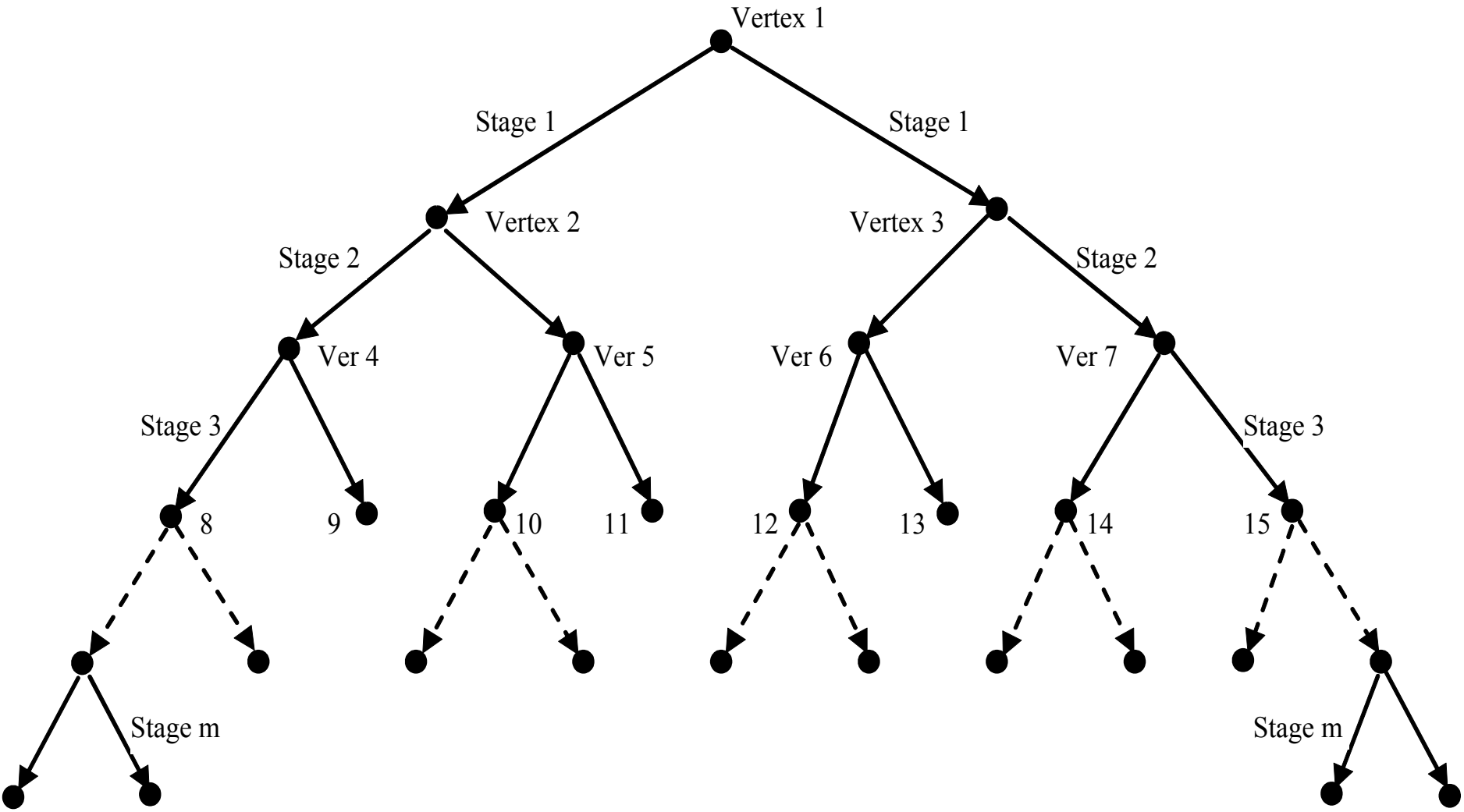
Pruning The Search tree

- Since in the search tree, a certain vertex represents that some arcs have been chosen and some other arcs have been eliminated, we can use SMST as a bound.
- At each vertex in the search tree, in addition to the feasibility checks, the current SMST is constructed and its total cost is calculated.
- If it is expensive than the cost of a solution already found, the sub-tree rooted at the current vertex in the search tree is pruned.

对搜索树剪枝

- 因搜索树中的一个顶点代表一些边被选上，而另一些边被移除，我们可以使用SMST的值作为一个界。
- 在搜索树的每个节点，除了可行性检查，当前SMST也被构造出来，并且计算出它的总花费。
- 如果该花费大于目前已求出的解的花费（上届），以当前节点为根的子树被剪枝。

The search tree



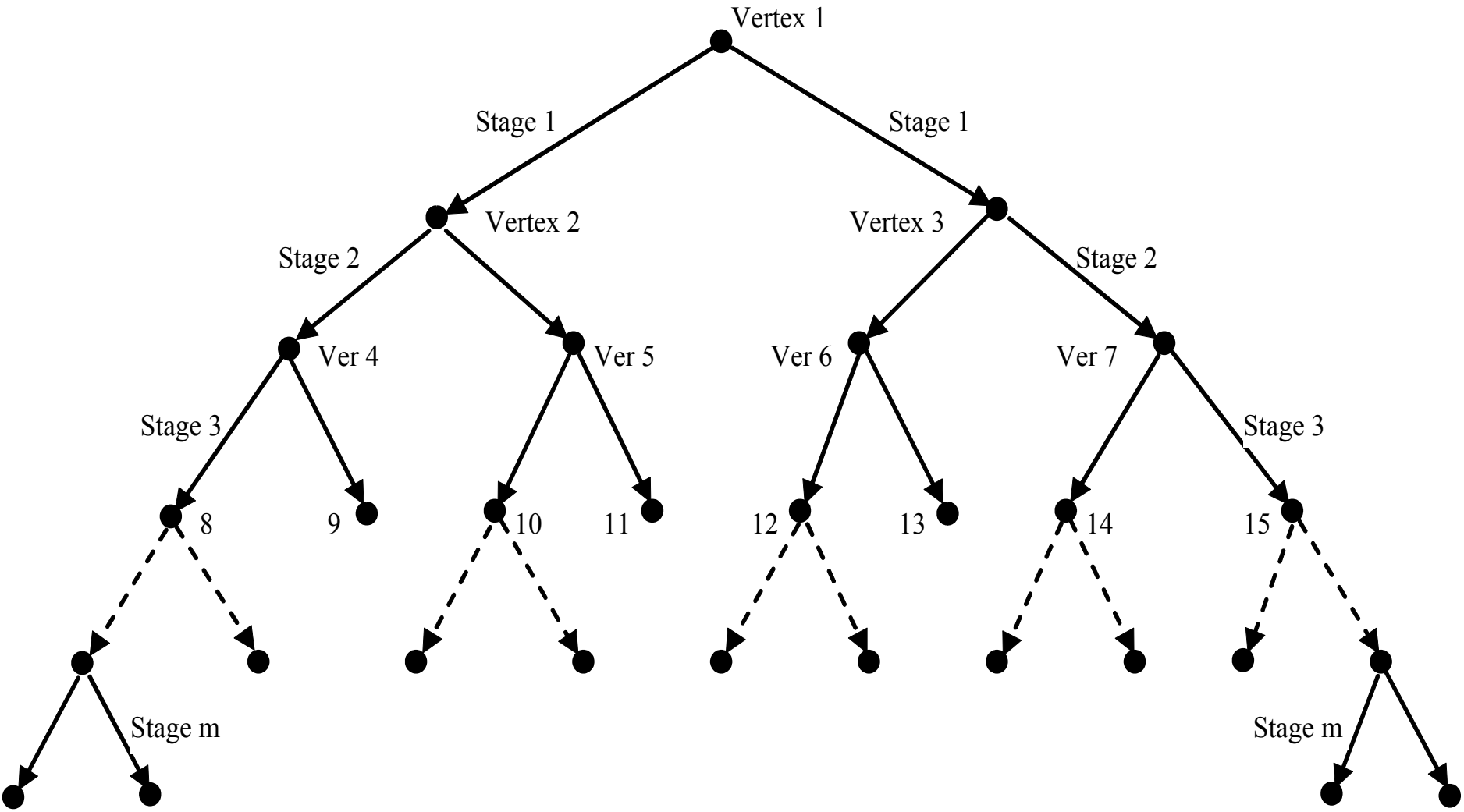
Pruning The Search tree

- Next?

Pruning The Search tree

- Next?
- Make the pruning faster.

The search tree



Pruning The Search tree ----feasibility

- In addition to the SMST bound, an efficient feasibility check procedure is greatly essential for pruning the search tree.

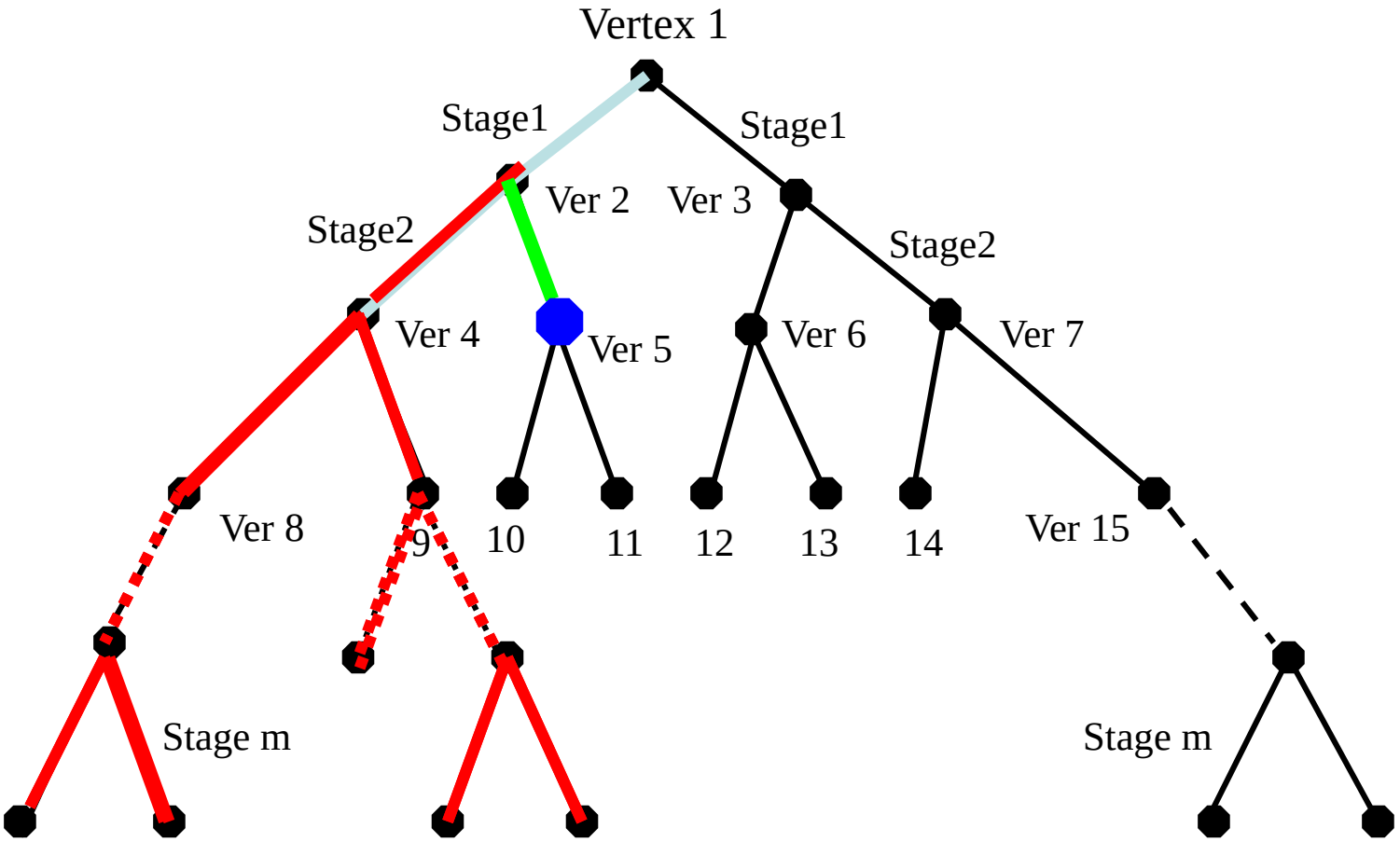
- When analyzing the above search routine, we found some shortcomings.
- When an arc, which would have a smaller arc number and would possess a higher stage in the search tree, has a very low flow capacity, it will often cause the weakness that our algorithm will spend a lot of time searching on the sub-tree under this stage while no feasible solution exists.

对搜索树剪枝---可行性

- 如果编号较小、在搜索中位于较高阶段的边的容量较小时，目前的算法会暴露出一个弱点：

  在该阶段花费大量时间搜索其子树，而实际上并不存在可行解。

Weakness— Arc 2 is not overflow at Ver. 4



What would you do?

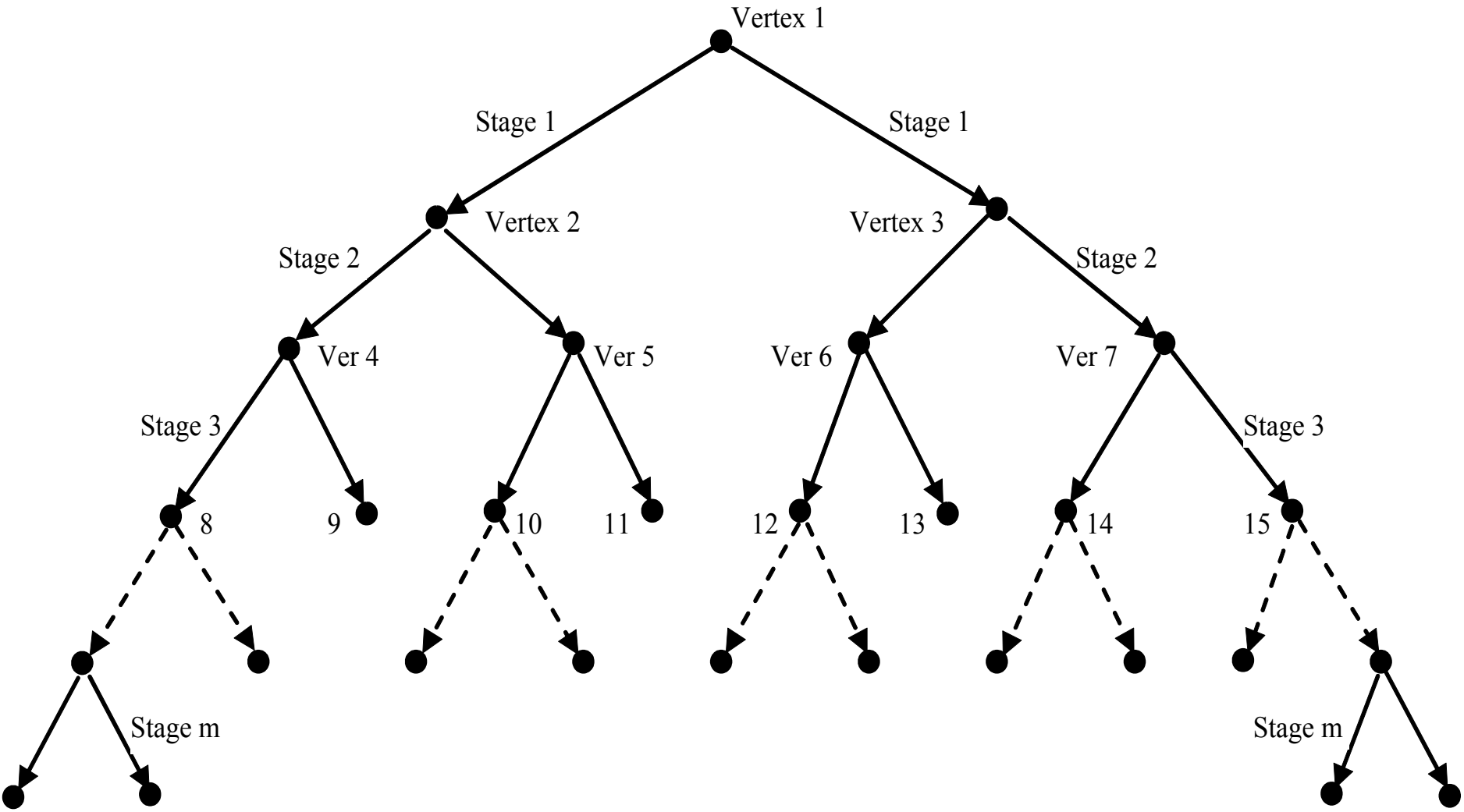
Predicting Violation of Capacity Constraint ！

- Any idea?

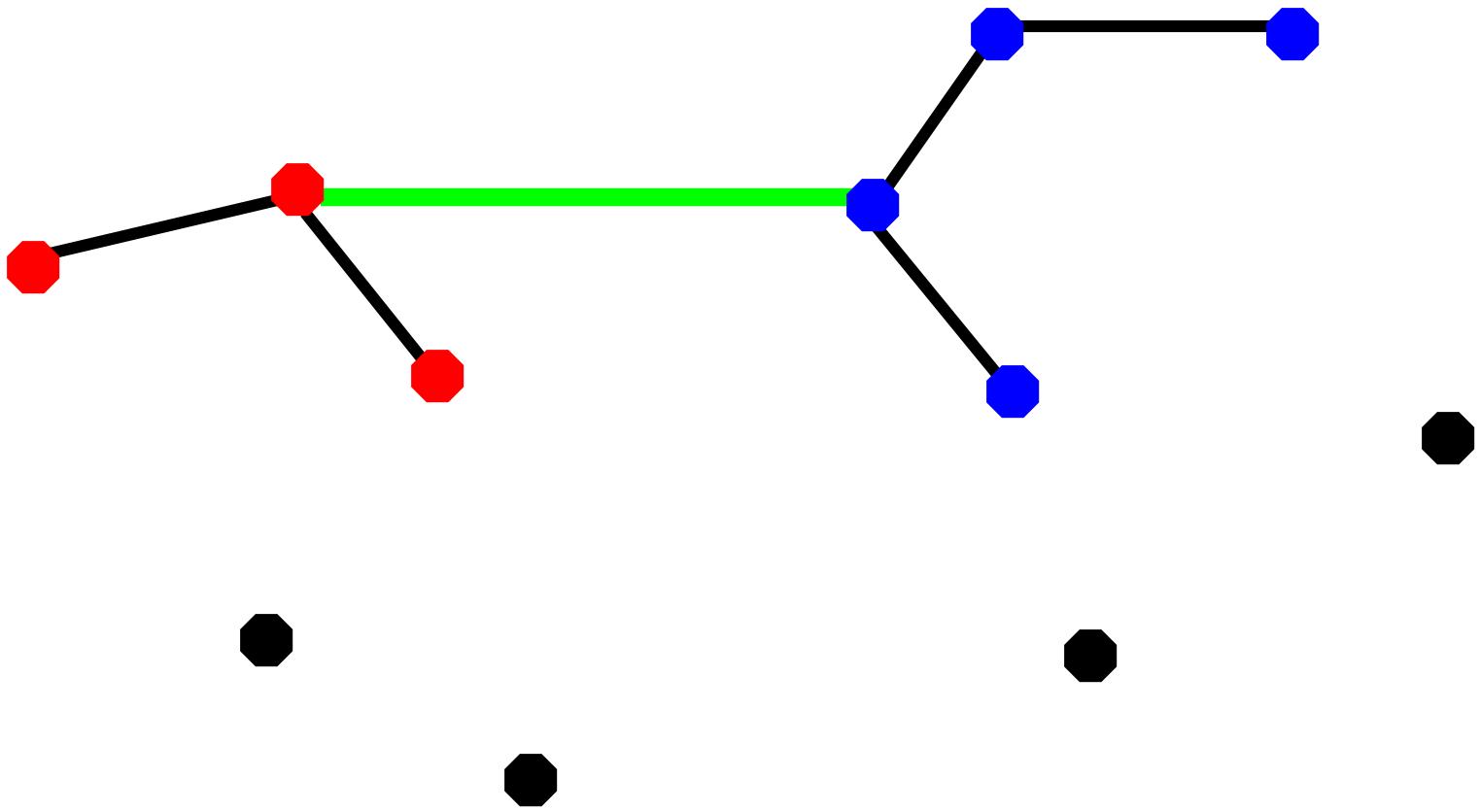
Predicting Violation of Capacity Constraint ！

- Any idea?
- No，but……

The search tree



Flow on an Arc



Predicting Violation of Capacity Constraint ！

- Any idea?
- Problem transformation

Predicting Violation of Capacity Constraint ！

- Any idea?
- Problem transformation
  - Unit traffic demand

consider a connected undirected graph $G=(V,A,d,c)$ with node set $V=\{1,2,...,n\}$ and arc set A.

There is a unit traffic demand $d_{ij}=1$ between each pair of nodes i and j in V. c_{ij} represents the cost of using arc (i,j) in A.

To satisfy the capacity constraint, the flow on any arc (i,j) must not exceed a given capacity k_{ij}. By means of these definitions, the generalized CMST problem is to find a minimum cost tree spanning the node set V where every arc (i,j) satisfies its capacity constraint k_{ij}.

Let f_{ij} denote the total flow on arc (i,j). Define $x_{ij}=1$, if arc (i,j) is included in the solution, and $x_{ij}=0$, otherwise.

Problem III

- Formulation-3

- Interpretation of the formulation

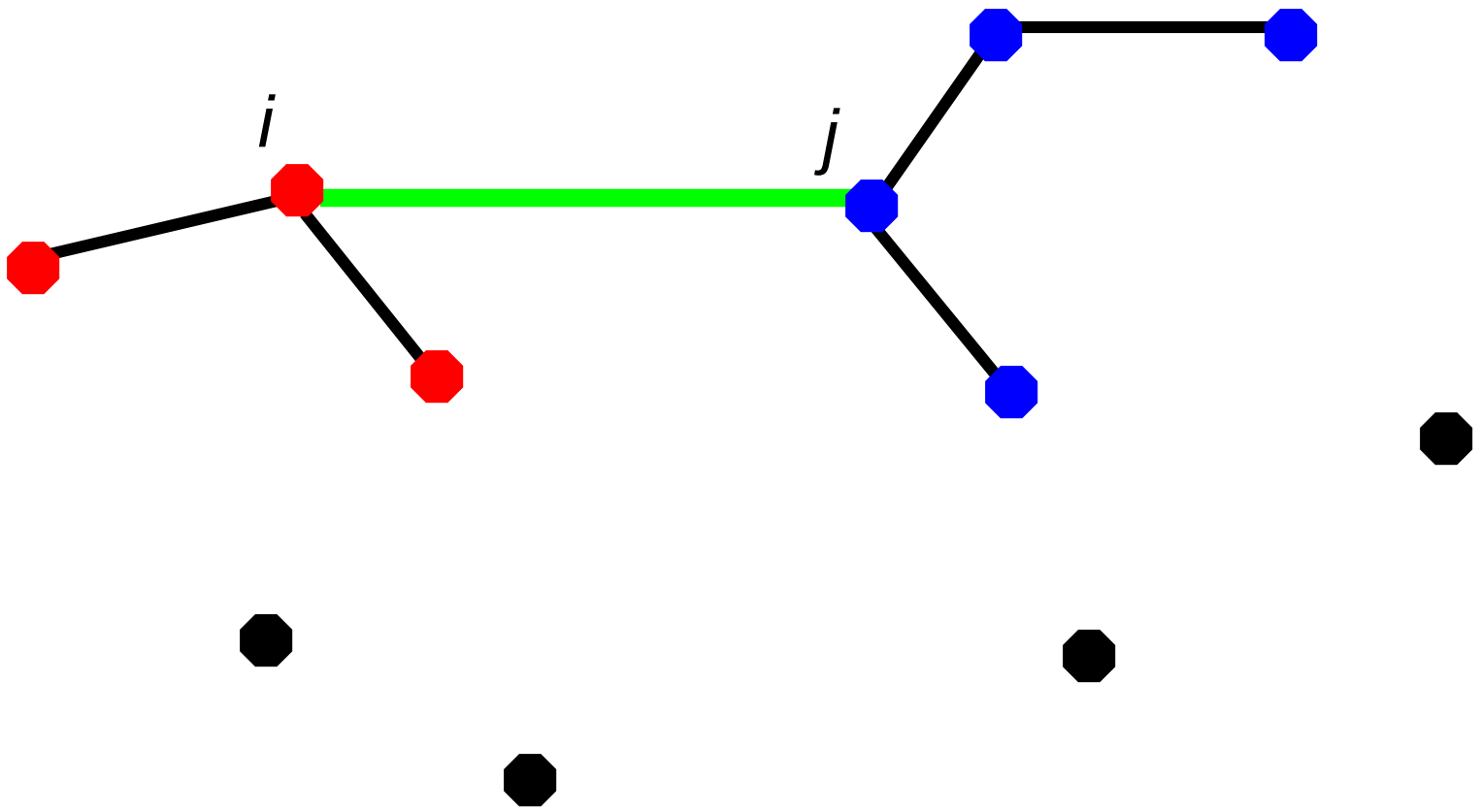
Predicting Violation of Capacity Constraint

While there is traffic demand between every pair of nodes, the flow on an arc can be calculated by just multiplying the number of nodes on one side of the arc and the number of nodes on the other side of the arc.

For any arc (i, j) in the solution tree, let V_1 be the maximal sub-tree containing node i but not containing node j; let V_2 be the maximal sub-tree containing node j but not containing node i.

Let NL be the number of nodes in V_1, NR be the number of nodes in V_2, then the flow on arc (i, j) must be $NL \times NR$ units.

Flow on an Arc



Predicting Violation of Capacity Constraint

While there is traffic demand between every pair of nodes, the flow on an arc can be calculated by just multiplying the number of nodes on one side of the arc and the number of nodes on the other side of the arc.

For any arc (i, j) in the solution tree, let V_1 be the maximal sub-tree containing node i but not containing node j; let V_2 be the maximal sub-tree containing node j but not containing node i.

Let NL be the number of nodes in V_1, NR be the number of nodes in V_2, then the flow on arc (i, j) must be $NL \times NR$ units.

Predicting Violation of Capacity Constraint

If arc (i, j) is in a complete feasible solution tree of an N node problem, then the sum of NL and NR must be equal to N.

That is, when $NL = 1$, NR must be $N - 1$; when $NL = 2$, NR must be $N - 2$, and accordingly.

Therefore, depending on the topology of the solution tree, the flow on arc (i, j) may be from $1 \times (N - 1) = N - 1$ units to $(N/2) \times (N/2) = N^2/4$ units (when N is a even number);

or from $N - 1$ units to $((N + 1)/2) \times ((N - 1)/2) = (N^2 - 1)/4$ units (when N is an odd number).

Predicting Violation of Capacity Constraint

Let $N_{\max}$ be the larger number between NL and NR, $N_{\min}$ be the smaller one, then the possible combinations of $N_{\max}$ and $N_{\min}$, and the value of flow on arc (i, j) should be as shown in the table.

N	Is	Even	N	is	odd
$N_{\min}$	$N_{\max}$	*Flow*	$N_{\min}$	$N_{\max}$	*Flow*
1	N-1	N-1	1	N-1	N-1
2	N-2	2(N-2)	2	N-2	2(N-2)
…	…	…	…	…	…
m	N-m	m(N-m)	m	N-m	m(N-m)
m+1	N-(m+1)	(m+1)(N-(m+1))	m+1	N-(m+1)	(m+1)(N-(m+1))
…	…	…	…	…	…
N/2	N/2	N*N/4	(N-1)/2	(N+1)/2	(N*N-1)/4

Predicting Violation of Capacity Constraint

It can be proved that, in the table, the value of flow is in an ascending order from top to bottom.

If arc (i, j) has been used to construct a feasible solution tree, and its $N_{\min}$ is m, then its flow capacity must be equal to or greater then m(N-m).

Moreover, if arc (i, j) is to be chosen to construct a feasible solution tree and its flow capacity is T, then, when m(N-m)<= T <(m+1)(N-m-1), the arc's $N_{\min}$ must be less than or equal to m, since otherwise, there will be an overflow on the arc.

Here we name the number m the *Link Degree* of arc (i, j).

To form a complete feasible solution, the Link Degree of arc (i, j) is actually the maximum number of nodes that can be included in the smaller one between the two end node sets (V_1 and V_2) of the arc.

Feasibility Bound

- Before the searching is engaged, the Link Degree value for every arc is calculated and stored.
- At every searching stage, even the current partial solution is feasible, the procedure checks if any arc's current NL and NR are both greater than its Link Degree.
- If so, the procedure performs backtracking and the subtree rooted at the current vertex in the search tree is pruned, since it will sooner or later cause an overflow if more arcs are added to the solution.

预测流量约束是否违背

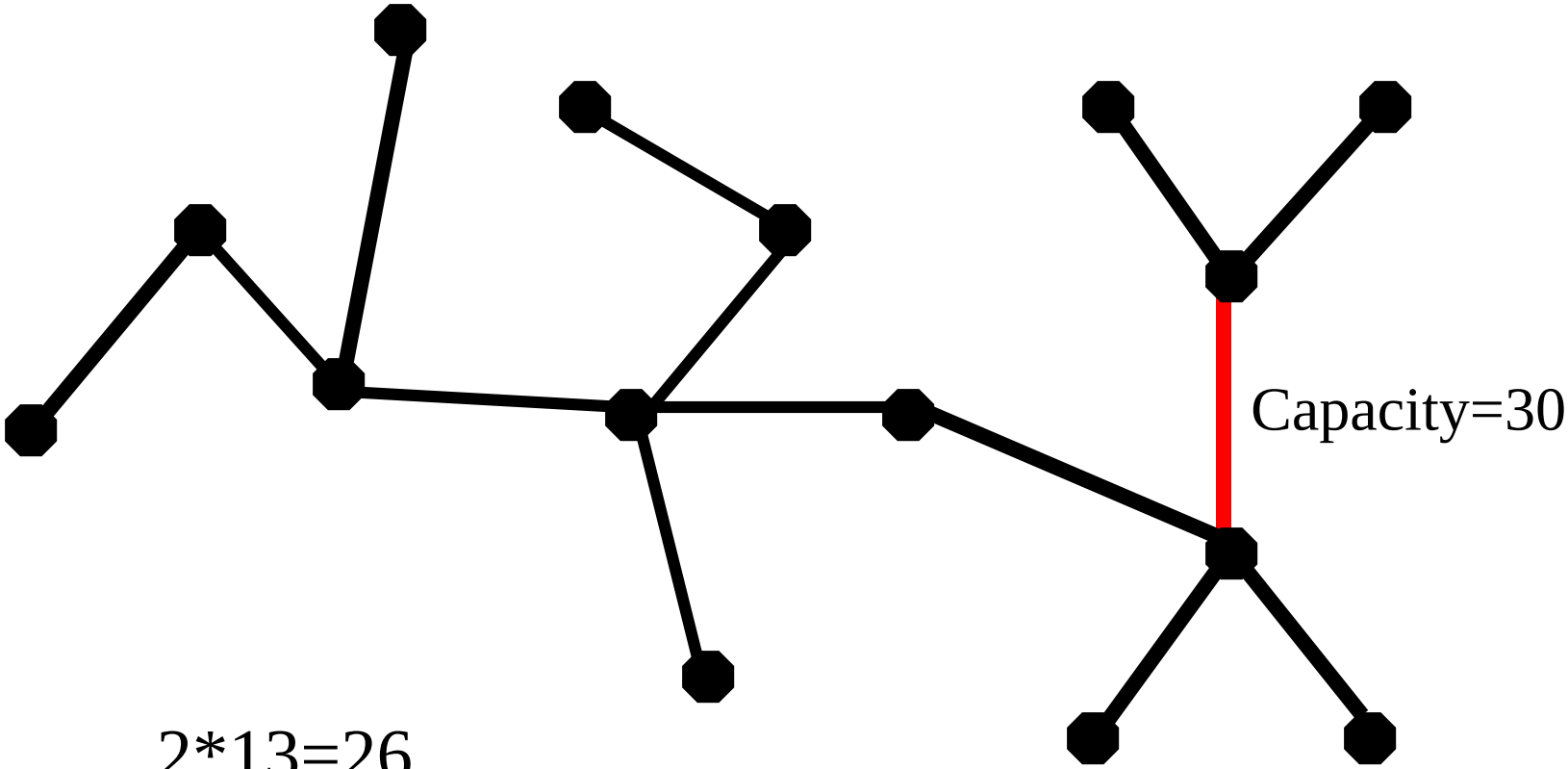
- 在搜索之前，每条边的Link Degree被计算并存储起来。
- 在搜索的每个阶段，即使当前部分解是可行的，算法要检查是否有某一条边的当前NL和NR都比它的Link Degree大。
- 如果是这样，回溯并且以当前顶点为根的子树被剪枝，因为随着越来越多的边加入到解集中，该边迟早会溢出。

Predicting Violation of Capacity Constraint

- Comparing the Link Degree and the current $N_{\min}$ can foretell whether an arc will overflow even if it hasn't yet been connected to some other separated arcs or sub-trees and hasn't yet overflowed.

- Implementing this technique will prune the search tree far more quickly.

Advantage



2*13=26

3*12=36

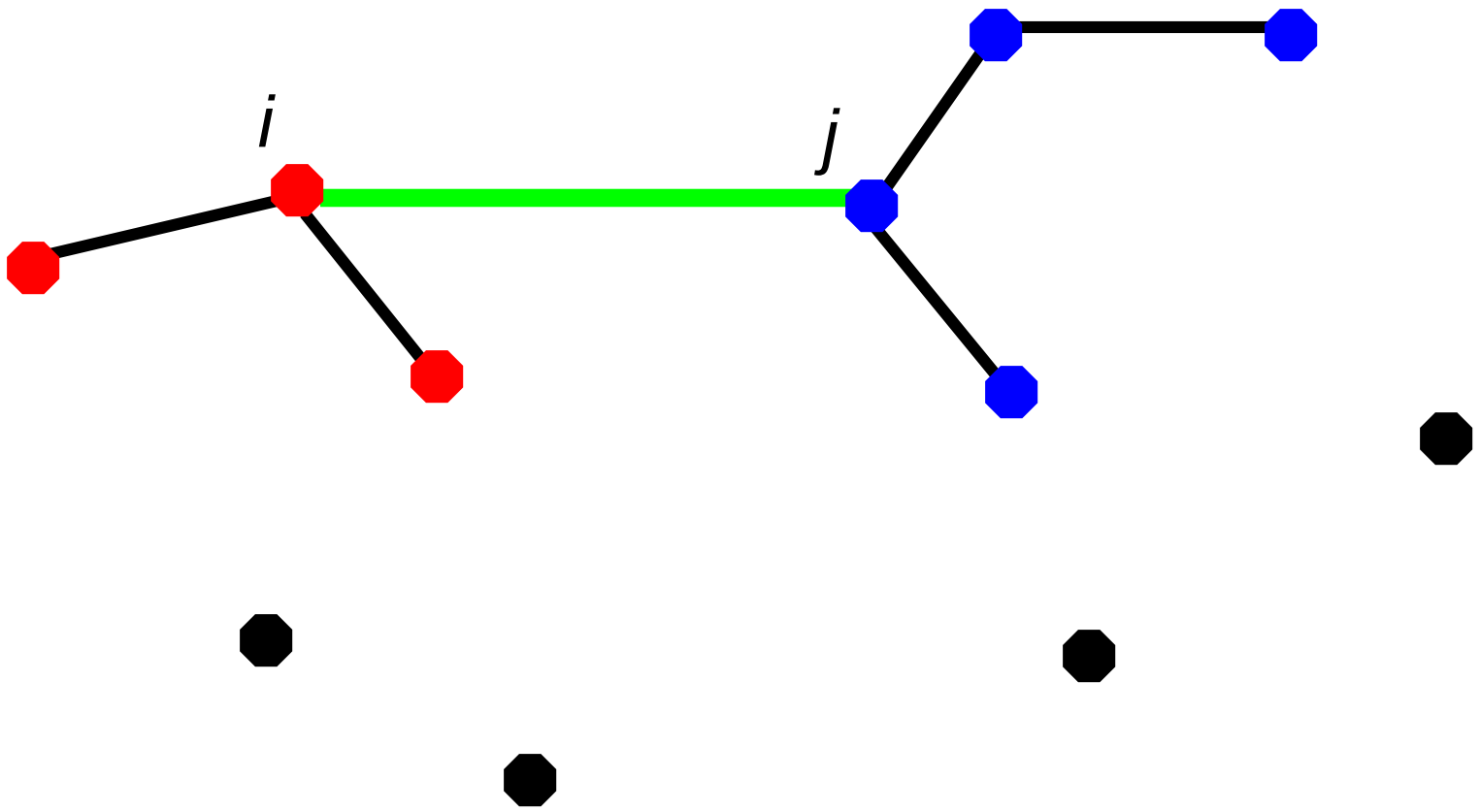
Feasibility Predicting Method for Problem with Non-Unit Traffic Demand

- Problem II
- What's the idea?

Feasibility Predicting Method for Problem with Non-Unit Traffic Demand

- Problem II
- What's the idea?
- All nodes will eventually be connected, either way !

Flow on an Arc



Predicting Violation of Capacity Constraint

At a certain searching stage, let F_c be the current traffic flow on arc (i, j), set V_l $(|V_l| = L)$ be the end node set of arc (i, j) containing node i, set V_r $(|V_r| = R)$ be the other end node set of arc (i, j) containing node j, set V_s $(|V_s| = S)$ be the set of nodes not having been connected to the arc.

Every node m in set V_s will be eventually connected to arc (i, j), and will be included in either V_l or V_r.

Predicting Violation of Capacity Constraint

Let FL_m be the total traffic demand between node m and all nodes in V_l, FR_m be the total traffic demand between node m and all nodes in V_r, that is,

$$FL_m = \sum_{I=1}^{L} Demand(m, V_l(i))$$

$$FR_m = \sum_{I=1}^{R} Demand(m, V_r(i)).$$

Then the minimum flow increased on arc (i, j) after connecting node m to the arc will be either FL_m (when V_l stay unchanged) or FR_m (when V_r stay unchanged). Let $F_{\min}^{m}$ be the smaller value between FL_m and FR_m, then the minimum flow increased on arc (i, j) after connecting all nodes in V_s to the arc is

$$F_{MIN} = \sum_{j=1}^{S} F_{\min}^{V_s(j)}.$$

Predicting Violation of Capacity Constraint

Let F_t be the minimum total flow on arc (i, j) after connecting all separated nodes to the arc, then $F_t = F_c + F_{MIN}$. If F_t is greater than the capacity of arc (i, j), we know that the arc must encounter overflow if a complete solution is formed.

At each searching stage, the F_t of every arc is calculated; if the F_t of any arc exceed its capacity, the procedure will perform backtracking. Here, F_t works like a bound and it helps the procedure prune the search tree greatly faster.

Future flow bound

- Can you make the bound tighter?

Future flow bound

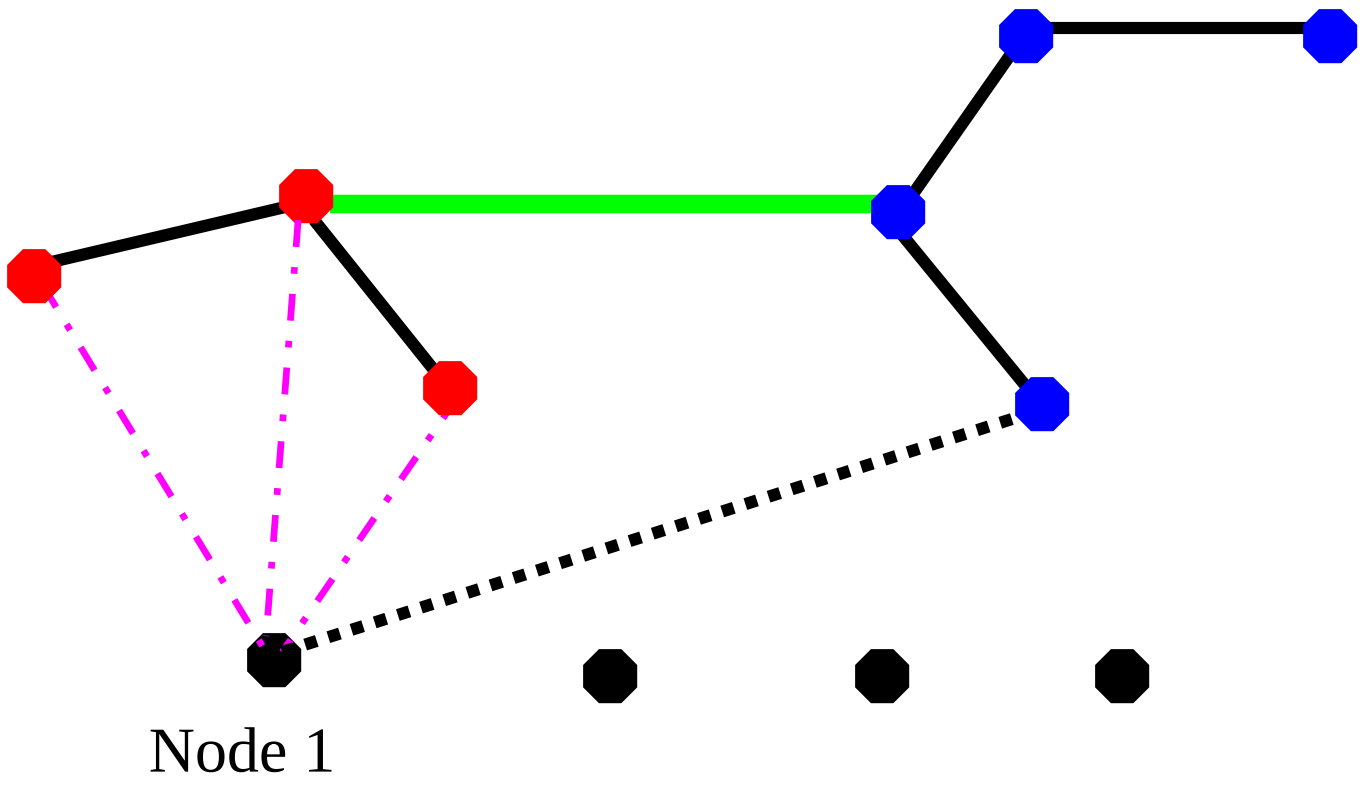
- Can you make the bound tighter?

- The F_t bound can be made even tighter if the possible minimum flow on an arc is calculated based on every separated sub-tree but not on single node.

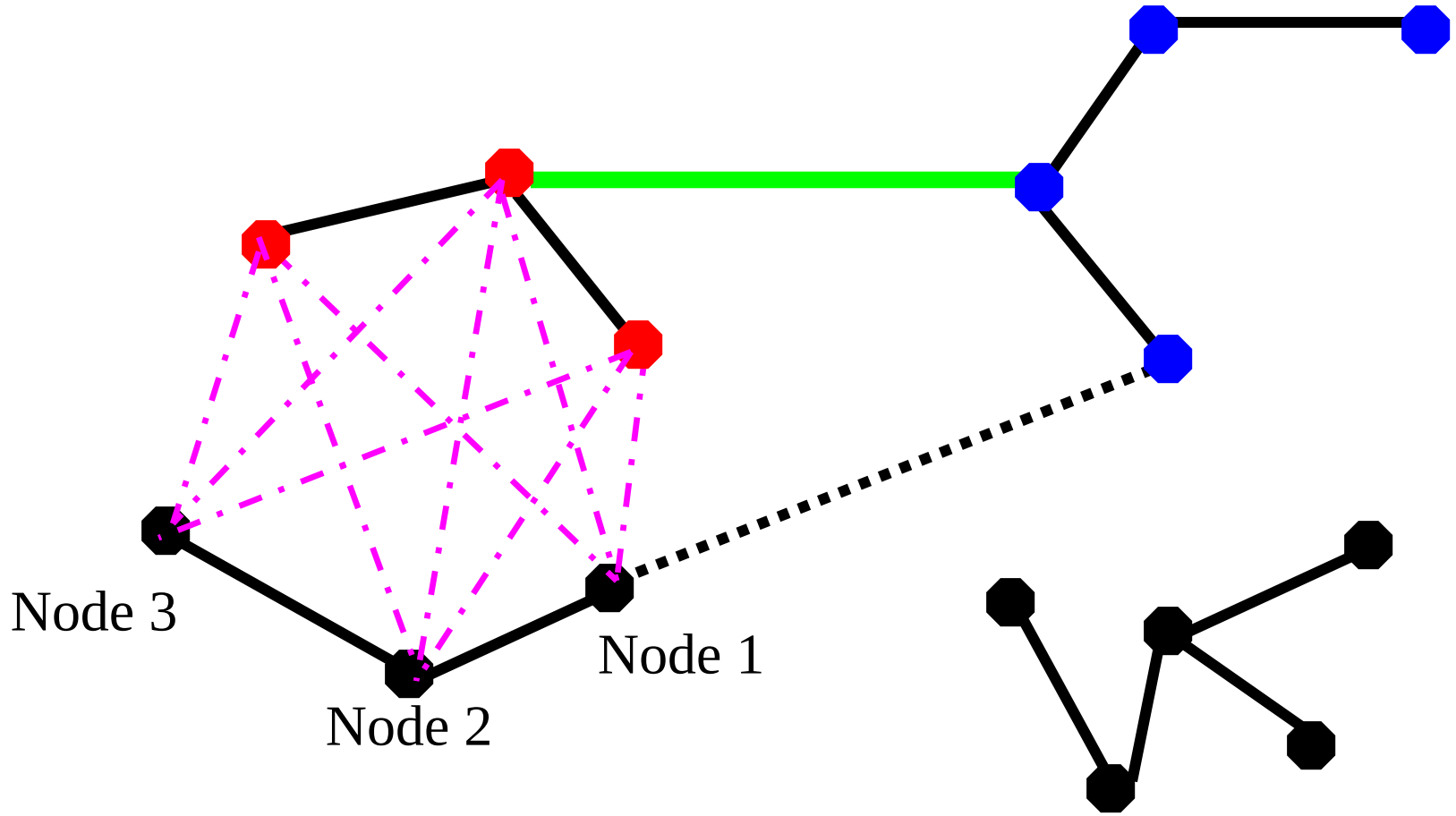
Tighter future flow bound for non-unit traffic demand problem

- The induction of tighter bound

Non-unit Demand Problem



Non-unit Demand Problem



Further improvement

- Can you make the searching more fast?

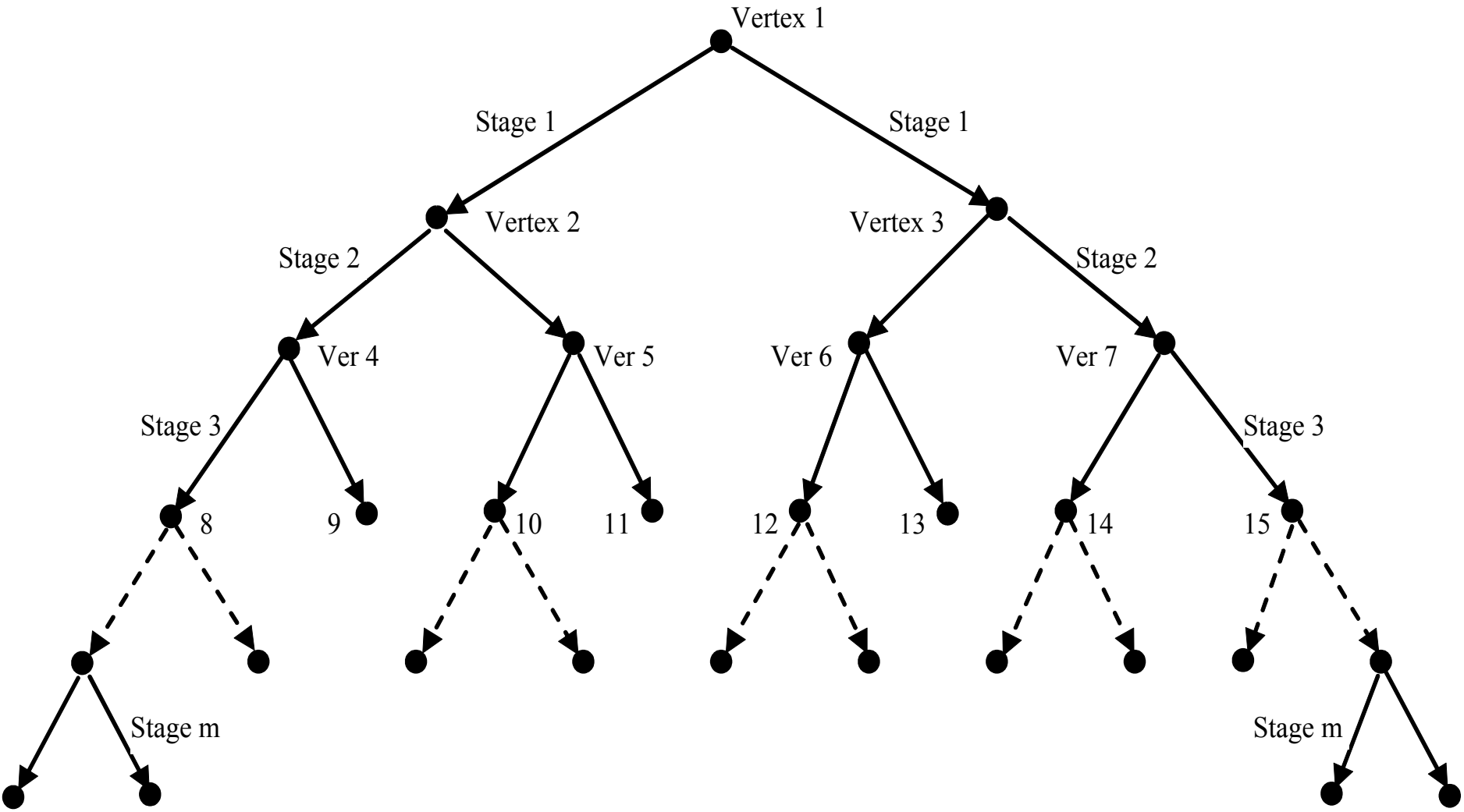
Further improvement

- Can you make the searching more fast?
- We sort the arcs in the order of ascendant arc cost and give each of them an arc number, from 1 to m, as identification.
- What’s the advantages?

Ascendant arc order

- Construction of SMST
- More advantages?

The search tree



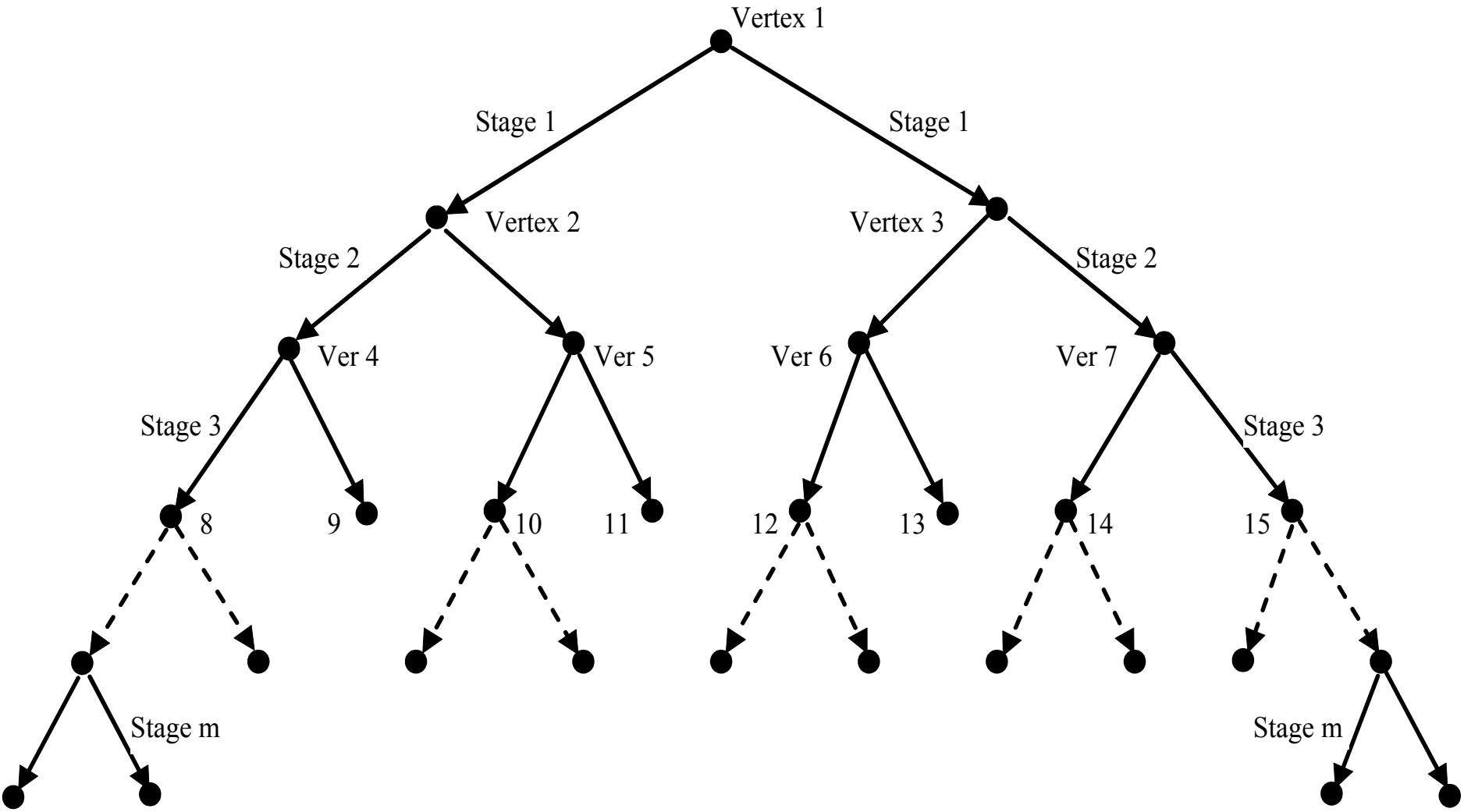
Ascendant arc order

- Construction of SMST
- More advantages?

Ascendant arc order

- Construction of SMST bound
- Dominant frontier search
- More advantages?

The search tree



Ascendant arc order

- Construction of SMST bound
- Dominant frontier search
- More advantages?

Ascendant arc order

- Construction of SMST bound
- Dominant frontier search
- Total cost bound

Further improvement

- Can you make the searching more fast?

Further improvement

- Can you make the searching more fast?
- An initial solution (upper bound)
  - A modified Prim's algorithm

The algorithm

- Data structures

- I consider the following to be the best data structure for an arc-orientated branch and bound searching.
  - define “SolutionBuffer”, which stores the arcs (identified by arc number) that have been chosen to form a potential feasible solution, as an array [1..N-1] of integer.
  - An integer “FlagSolve” is used to store the total number of arcs already been added to “SolutionBuffer” at the current searching step.

The algorithm

- Data structures

SolutionBuffere= [1, 3, 5, 6, 9, … , x]

- I consider the following to be the best data structure for an arc-orientated branch and bound searching.
  - define “SolutionBuffer”, which stores the arcs (identified by arc number) that have been chosen to form a potential feasible solution, as an array [1..N-1] of integer.
  - An integer “FlagSolve” is used to store the total number of arcs already been added to “SolutionBuffer” at the current searching step.

The algorithm

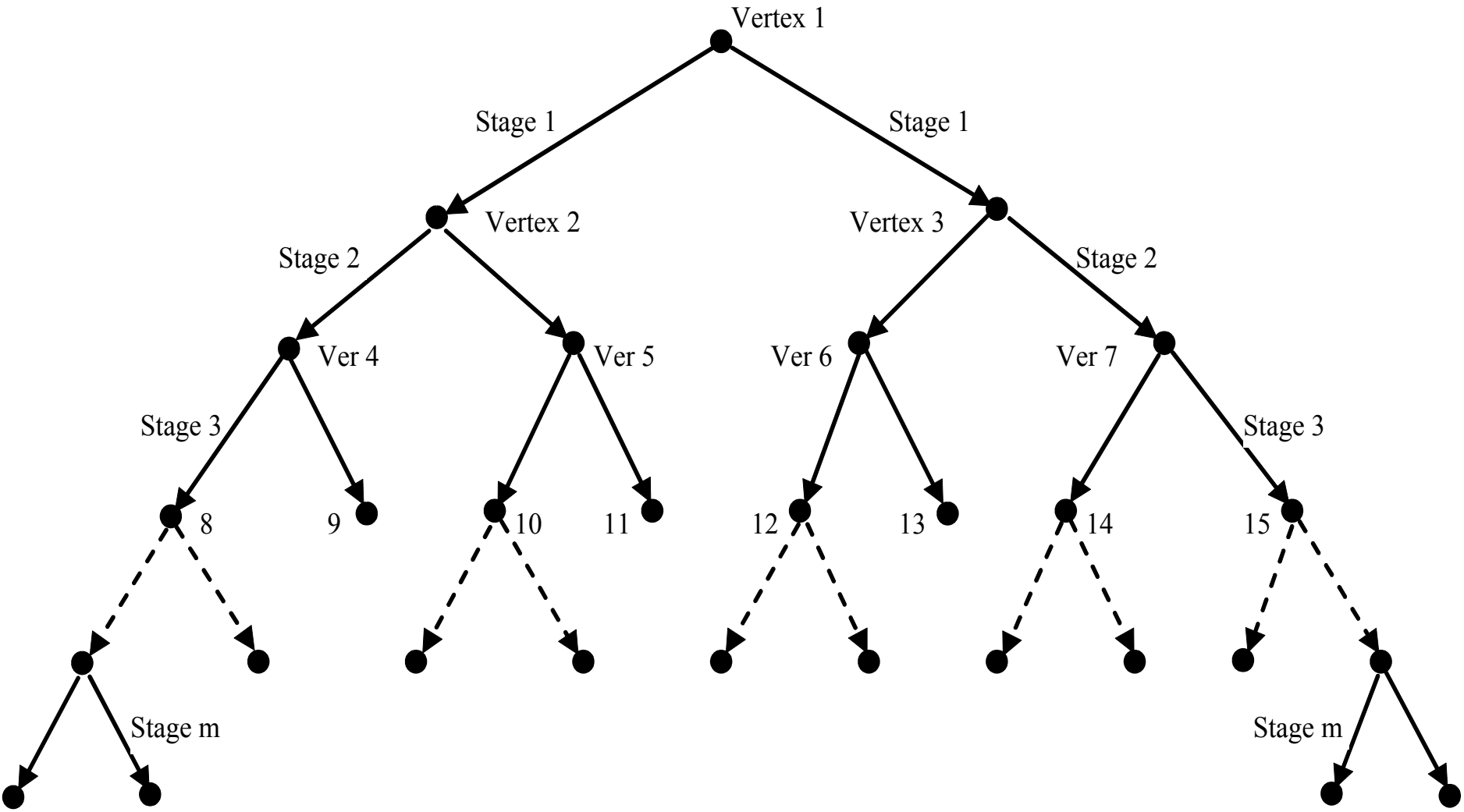
- Codes

Faster ?

Faster ?

Faster forward traversing guided by SMST

The search tree

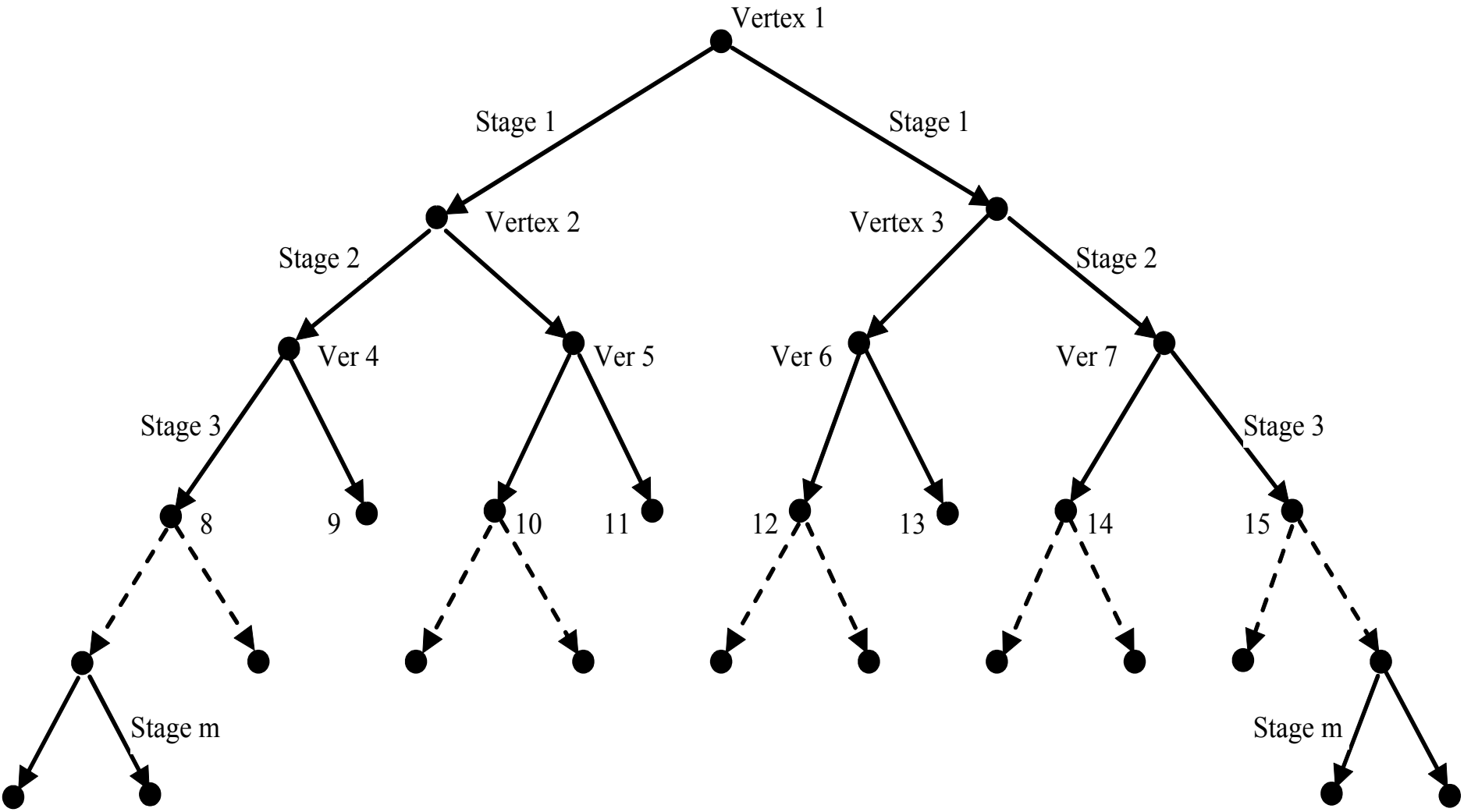


Faster ?

Faster ?

Omitted SMST Construction

The search tree

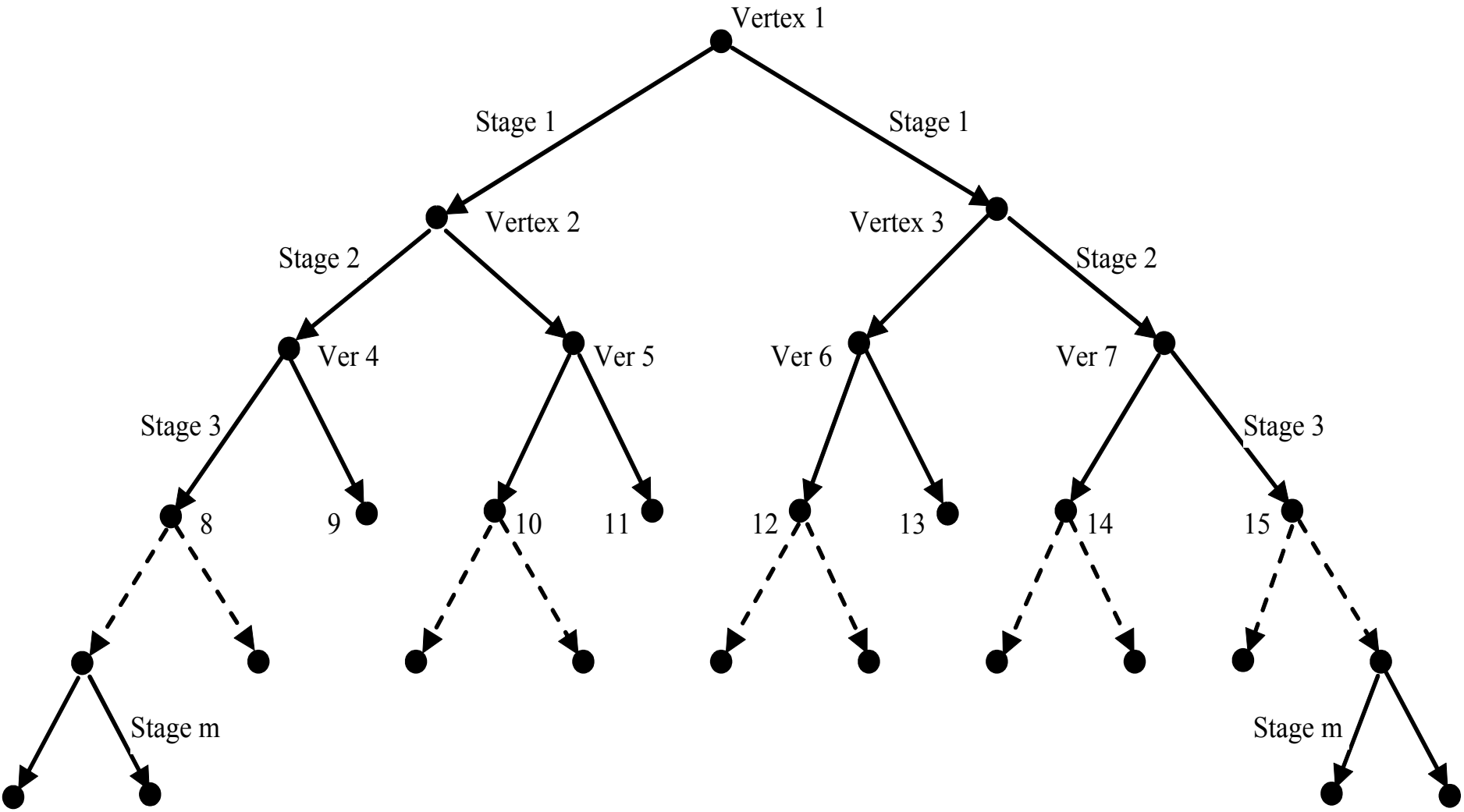


Faster ?

Faster ?

Omitted cycle detection

The search tree



Faster ?

Faster ?

Sequence of function calls

Problem I

- Non-linear cost function
- What's your idea?

Problem I

- Non-linear cost function
- What's your idea?
- Piecewise constant

Problem I

- Non-linear cost function
- What's your idea?
- Piecewise constant
- What would you do?

The efficiency of the algorithm

Problem File Name	Running Steps	Running Time (Seconds)
P100g-1.txt	169474	163
P100g-2.txt	279842	407
P100g-3.txt	156281	140
P100g-4.txt	188611	160
P100g-5.txt	3312304	720
P100g-6.txt	98158	182
P100g-7.txt	217784	400
P100g-8.txt	140683	163
P100g-9.txt	628903	700
P100g-10.txt	5804	8

NP - *Completeness*

NP - *Completeness*

Why the theories ?

- One of the primary purposes of scientific inquiry is to help structure the world around us so that we can better understand it.
  - The periodic chart of elements
  - The genus/species nomenclature
  - Discrete/continuous mathematics

- Is there a way to develop a structural understanding of algorithms or for the problems to which we wish to apply them?
  - Computational complexity theory in early 1970s

Why the theories ?

The theory of *NP*-completeness helps us to classify a given problem into two broad classes:

(1) easy problems that can be solved by polynomial-time algorithms, and

(2) hard problems that are not likely to be solved in polynomial time and for which all known algorithms require exponential running time.

NP Completeness

- Notice that in this classification we want to determine only whether a problem can or cannot be solved in polynomial time;

  the order of the polynomial is irrelevant.

- Keep this point in mind throughout our subsequent discussion.

Easy or Hard ?

- We call a class of problems **Easy** if we can develop an algorithm to solve every instance of the problem class in **polynomial time**.
  - A polynomial-time algorithm is referred to as an ***efficient algorithm***
- We have not developed ***efficient algorithms*** for many problems.
- Unsuccessful attempts have led us to question whether these problems are ***inherently hard*** in the sense that no ***efficient algorithm*** could possibly ever solve these problems.

NP Completeness

- The theory of NP-completeness is an outgrowth of these inquiries.
- Although this theory has been unable to prove that these difficult problems admit no efficient algorithms, the theory has shown that the majority of these problems are equivalent to each other in the sense that if we could develop an efficient algorithm for one problem in this class, we would then be able to develop an efficient algorithm for *every* other problem in this class.
- We refer to this broad class of "computationally equivalent" problems as NP-complete problems.

NP Completeness

- The theory of NP-completeness is an outgrowth of these inquiries.
- 尽管该理论未能证明确实不存在求解这些难问题的高效算法，但是却已证明这些难问题大多数是互相等价的，即：如果我们可以找到针对某一个这类问题的高效算法，那么我们就能找到求解该类问题中其他所有问题的高效算法。
- 我们把这类“计算上等价”的问题称为NP-complete问题.

NP Completeness

- This class now includes thousands of problems and possesses the remarkable property (which is somewhat difficult to believe initially) that each problem in this class can be transformed to every other problem in polynomial time.

- As a consequence, each problem is "just as hard" as every other problem.
- This relationship suggests that NP-complete problems share some generic difficulty that is beyond the reach of polynomial-time algorithms.
- Indeed, the research community widely believes that NP-complete problems *cannot* be solved efficiently.

Bad news ***vs.*** Saving your job

The theory of NP-completeness also has its positive aspects.

- You are intelligent enough not to return to your boss's office and report, "***I can't find an efficient algorithm, I guess I am just too dumb.***"

- Proving its inherent difficulty could be as difficult as finding an efficient algorithm. You cannot walk into your boss's office and declare, "***I can't find an efficient algorithm because no such algorithm is possible!***"

- Just as hard as a large number of other problems that have defied solution by an efficient algorithm despite decades of efforts of the brightest researchers.

  Then you can confidently march into your boss's office and announce, " ***I can't find an efficient algorithm, but neither can these famous people***."

Utility in practice

- Whenever we encounter a new problem of some practical or theoretical interest, we try to develop an efficient algorithm for solving it.

- If we do succeed, clearly the problem is easy and we and others might make further attempts to develop an even more efficient algorithm.

- However, if we do not succeed in developing an efficient algorithm for the problem, we might begin to wonder whether our problem is an NP-complete problem.

Utility in practice

- The theory of NP-completeness provides us with several tools for establishing that a problem is NP-complete.
- If we do succeed in showing the problem is NP-complete, we have sufficient reason to believe that the problem is hard and no efficient algorithm can ever be developed to solve it.
- We should thus abandon our quest for an efficient algorithm and direct our efforts at developing efficient heuristics or

  at developing various types of enumeration algorithms or other algorithms that will generally run in exponential time.

Heuristic

A heuristic is

"a technique which seeks **good** (i.e. near optimal) solutions at a **reasonable computational cost**

without being able to guarantee either feasibility or optimality,

or even in many cases to state how close to optimality a particular feasible solution is"

– Reeves and Beasley