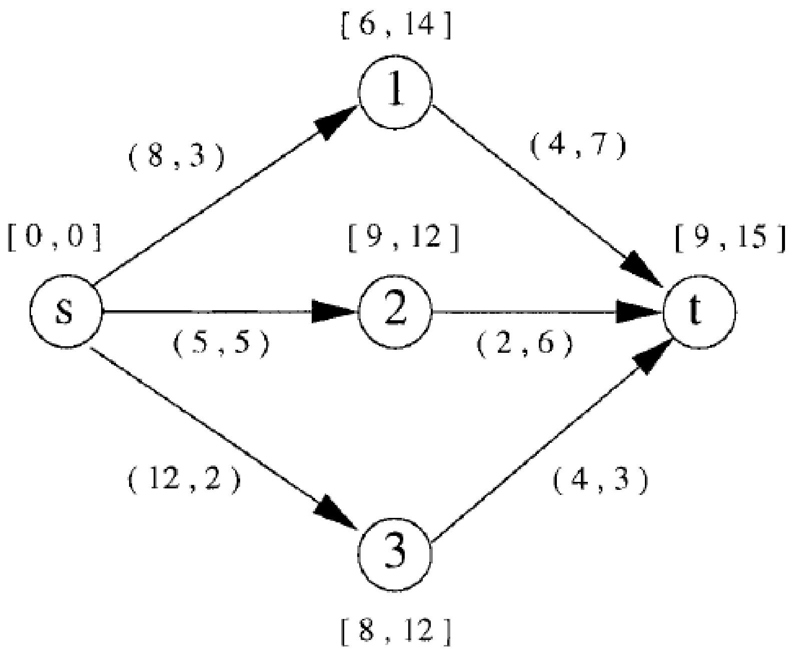
Labeling Algorithm

Su E

July 17, 2020

SPPRC (see 《Column Generation》. Chapter 2)

Shortest Path problem with resource constraints



$G = \{V, A\}$

- $V = \{s, 1,2, \ldots, n, t\}$ is non-empty set of nodes
- A is non-empty set of arcs
- node i has time window $[a_i, b_i]$
- $\mathrm{arc}(i, j)$ is a two-dimensional vector
  - 1st component t_{ij} provides the travel time
  - 2nd component c_{ij} indicates the cost

Objective

- generate a minimum cost path from s to t
  - satisfy all the resource constraints

Algorithm Overview

Dynamic programming (DP)

- start from the path $p_s = \{s\}$
- use resource extend functions (REFs) to extend paths one-by-one into all feasible directions to obtain partial paths $p_i = \{s, v_1, \dots, v_i\}$, which are encoded by labels
- efficiency depends on the ability to identify and discard paths which are not useful to build a Pareto-optimal set of paths
- discarding non-useful paths is achieved by a dominance rules which strongly depend on the path-structural constraints and the properties of the (REFs)

Label Definition

A label $L_i = \{v, c_i, q, t\}$ is a tuple representing a partial path from the origin depot s to a vertex i

- v is the last visited vertex in the path, i.e. $v = i$
- c_i is the reduced cost of the partial path
- q is cumulated load along the path
- t is the earliest time at which service can start at vertex v

q and t are so-called resource variables

```java
public class Label implements Comparable<Object>, Cloneable {
    //----------------------------
    //----state information--------
    //----------------------------
    public int id;
    public double c;
    public double s;
    public int l;
    //-----------------------------
    //--other information----------
    //------------------------------
    public Label father;
    public Data data;

    public Label(Data d){
        data = d;
    };

    public int compareTo(Object o){
        Label l=(Label) o;
         if(c>l.c)
            return 1;
         else if(c==l.c)
            return 0;
         else
            return -1;
    }
```

Extension Function

- Let $L_s = (s, 0, 0, 0)$ be the initial label at vertex s
- Denote by $v(L), c(L), q(L), t(L)$ the components of a label L associated with a path $p(L)$ ending at vertex $v(L)$.
- Extend L along an arc (i, j), a new label L' is obtained by applying the following relations:
  - $v(L') = j$
  - $c(L')= c(L) + c_{ij}$
  - $q(L')= q(L) + q_j$
  - $t(L')= \max\{t(L) + s_i + t_{ij}, a_j\}$

L' represents path $\mathrm{p}(L') = p(L) \oplus (i, j)$

- $\mathrm{p}(L')$ is feasible if $q(L') \in [0, Q]$ and $t(L') \in [a_j, b_j]$, otherwise, it is infeasible, and label L' is discarded.

Dominance Rule

- Feasible extension set of $p(L)$:

$$\mathcal{E}(L) = \{w\text{: } p(L)\oplus w \text{ } is \text{ } a \text{ } feasible \text{ } path \text{ } and \text{ } w = \{v(L), \dots, t\} is \text{ } a \text{ } partial \text{ } path\}$$

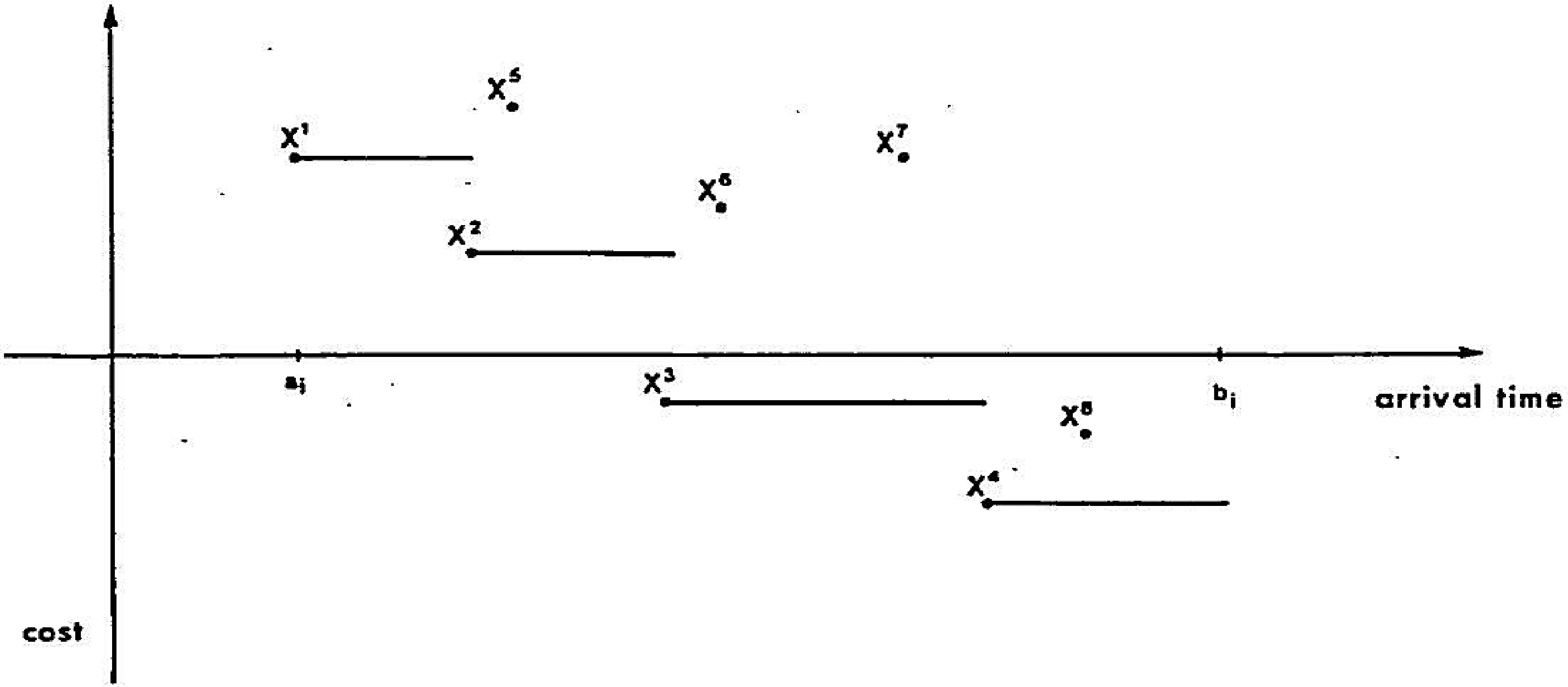
- A label L can be discarded if there exists a set of labels $\mathcal{L} = \{L_1, \dots, L_{|\mathcal{L}|}\}$, s.t. for all feasible extensions $w \in \mathcal{E}(L')$, there exists a label $L \in \mathcal{L}$ for which
  - $p(L)\oplus w$ is feasible
  - $c(p(L)\oplus w) \leq c(p(L')\oplus w)$
- A label L dominates label L' if
  - $v(L) = v(L')$
  - $c(L) \leq c(L')$
  - $\mathcal{E}(L) \supseteq \mathcal{E}(L')$

Sufficient Conditions

- Let $(R_i{}^1, C_i{}^1)$ and $(R_i{}^2, C_i{}^2)$ be two different labels for to paths from s to i. Then Label L_1 dominates L_2, i.e. $(R_i{}^1, C_i{}^1) \prec (R_i{}^2, C_i{}^2)$ if and only if $(R_i{}^2, C_i{}^2) - (R_i{}^1, C_i{}^1) \geq 0$
- For all resource extension functions are non-decreasing, then Label L dominates L' if
  - $v(L) = v(L')$
  - $c(L) \leq c(L')$
  - $r(L) \leq \mathrm{r}(L'), for\ any\ r \in Resources$

```
/**  dominance rules  */
public boolean dominate(Label label){
    if(id != label.id)
        return false;
    if(s > label.s)
        return false;
    if(l > label.l)
        return false;
    if(c > label.c)
        return false;
    return true;
}
```

Dominance Example



Label setting and label correcting algorithms (see Desrosiers .1995)

A label setting algorithm for SPPTW

Step 0. Initialization

$Q_o = \{(T_o^1 = a_o, C_o^1 = 0)\}$; $Q_i = \emptyset, \quad \forall i \in N \cup \{d\}$;

$P_i = \emptyset, \quad \forall i \in V.$

Step 1. Selection of the next label to be treated

Choose a label (T_i^k, C_i^k) with minimal T_i^k from $\bigcup_{i \in V}(Q_i \setminus P_i)$;

If $\bigcup_{i \in V}(Q_i \setminus P_i) = \emptyset$ then STOP.

Step 2. Treatment of label (T_i^k, C_i^k)

For all $j \in \Gamma(i)$,

$Q_j := \mathrm{EFF}\left(f_{ij}(T_i^k, C_i^k) \cup Q_j\right)$;

$P_i := P_i \cup \left\{(T_i^k, C_i^k)\right\}$;

Return to *Step 1.*

A label correcting algorithm for SPPTW

Step 0. Initialization

$Q_o = \left\{(T_o^1 = a_o, C_o^1 = 0)\right\}$;

$Q_i = \left\{(T_i^1 = a_i, C_i^1 = \infty)\right\} \quad \forall i \in V \setminus \{o\}$; $\mathcal{L} = \{o\}$.

Step 1. Treatment of node i

Choose a node $i \in \mathcal{L}$;

For all $j \in \Gamma(i)$ do:

$Q_j' = \mathrm{EFF}\left(\bigcup_k f_{ij}(T_i^k, C_i^k) \cup Q_j\right)$,

If $Q_j' \neq Q_j$ then $Q_j = Q_j'$ and $\mathcal{L} := \mathcal{L} \cup \{j\}$.

Step 2. Reduction of $\mathcal{L}$

$\mathcal{L} := \mathcal{L} \setminus \{i\}$;

If $\mathcal{L} = \phi$ then STOP, otherwise return to *Step 1*.

CODE (partial)

```java
ArrayList<Integer> neg_index = new ArrayList<Integer>();
//initialize the UL, TL
if(UL.isEmpty() == false){
    UL.clear();
}
if(TL.isEmpty() == false)
    TL.clear();
for(int i = 0; i < data.N; i++){
    UL.add(new ArrayList<Label>());
    TL.add(new ArrayList<Label>());
}
// add the first label
Label fl = new Label(data);
fl.id = 0;
fl.c = - lp.mu[0];
fl.l = data.q[0];
fl.s = data.e[0];
fl.father = null;
UL.get(0).add(fl);
```

```java
while(neg_index.size() < size){
    Label label = null;
    //---------------------------------------------
    //--------find a label to extend---------------
    //---------------------------------------------
    for(int i = 0; i < UL.size(); i++){
        if(UL.get(i).size() > 0){
            label = UL.get(i).get(0);
            UL.get(i).remove(0);
            break;
        }
    }
    if(label == null)
        break;
    //---------------------------------------------
    //----------add the TL-------------------------
    //---------------------------------------------
    TL.get(label.id).add(label);
```

```java
for(int i = 1; i < data.N + 1;i++){
    //------------------------------------------------------
    //----------test the feasibility of extension-----------
    //------------------------------------------------------
    if(lp.node.branch_arcs[label.id][i] == -1)
        continue;

    double edge_profit = lp.dual_branch[label.id][i] + lp.mu[i] +

    if(label.l + data.q[i] <= data.Q && label.s + data.st[label.i
        Label new_label = new Label(data);
        new_label.id = i;
        new_label.c = label.c  + (data.ck[0] * label.l + data.fk
        - edge_profit;
        if(i < data.N)
            new_label.s = Math.max(data.e[i], label.s + data.st[l
        else
            new_label.s = label.s + data.st[label.id] + data.t[la
        new_label.l = label.l + data.q[i];
        new_label.father = label;
        //---------------------------------------------------
        //---------add to the queue--------------------------
        //---------------------------------------------------
        if(i < data.N && dominate(new_label) == false)
            UL.get(new_label.id).add(new_label);
        if(i == data.N && new_label.c < -0.0000001){
            get_index(lp, new_label, neg_index);
        }
    }
```

Domination Test

```
/**  domination test  */
public boolean dominate(Label l ){
    ArrayList<Label> q = TL.get(l.id); // 同一id的extended label集合
    if(q!=null){
        for(int i = 0; i < q.size(); i++)
            if(q.get(i).dominate(l) == true)
                return true;
    }
    q = UL.get(l.id); // 同一id的unextended label集合
    if(q!=null){
        for(int i = 0; i < q.size(); i++)
            if(q.get(i).dominate(l) == true)
                return true;
    }
    // anti dominate
    if(q != null){
        for(int i = 0; i < q.size(); i++){
            if(l.dominate(q.get(i)) == true){
                q.remove(i);
                i--;
            }
        }
    }
    return false;
}
```

Introduce resource variables to solve ESPPRC (see Feillet .2004)

A customer resource vector

$$\mathcal{N}(L) = \{i\text{: } customer\ i\ is\ visited\ along\ the\ path\ L\}$$

Dominance rule include

$$\mathcal{N}(L) \subseteq \mathcal{N}(L')$$

$$\Downarrow$$

A unreachable customer resource vector

$$\mathcal{U}(L) = \{i\text{: } customer\ i\ is\ visited\ along\ the\ path\ L\ or\ becomes\ unreachable\ due\ to\ the\ resource\ constraints\}$$

Dominance rule include

$$\mathcal{U}(L) \subseteq \mathcal{U}(L')$$

Bidirectional Labeling (see Righini. 2006, 2008)

- Labels are extended both forward from s to its successors and backward from t to its predecessors
- Define the backward label $L_i = \{v, c_i, q, t\}$
  - v is the last visited vertex in the path, i.e. $v = i$
  - c_i is the reduced cost of the partial path
  - q is cumulated load along the path
  - t is the latest time at which service can start at vertex v
- Backward extension function, L_i along the $arc(j, i)$ to generate a new label L'_j:
  - $v(L') = j$
  - $c(L') = c(L) + c_{ji}$
  - $q(L') = q(L) + q_j$
  - $t(L') = \min\{t(L) - s_j - t_{ji}, b_j\}$

Bidirectional Labeling (see Righini. 2006, 2008)

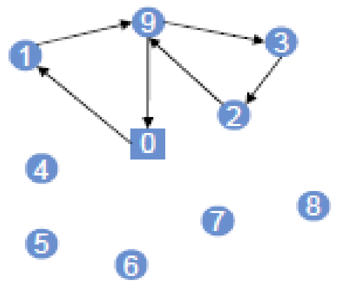
- Backward dominance rule: L dominates L' if
  - $v(L) = v(L')$
  - $c(L) \leq c(L')$
  - $r(L) \leq \mathrm{r}(L'), for\ any\ r \in Resources$
- Resource-based bounding
  - Forward - $t(L)$ > T/2
  - Backward - $t(L)$ < T/2
- A forward path $(i, c^{fw}, q^{fw}, t^{fw}, {n_k}^{fw})$ $\oplus$ a backward path $(j, c^{bw}, q^{bw}, t^{bw}, {n_k}^{bw})$
  - $c = c^{fw} + c^{fw} + c_{ij}$

  Feasibility test
  - $q^{fw} + q^{bw} \leq Q$
  - $t^{fw} + s_i + t_{\mathrm{ij}} \leq t^{bw}$
  - ${v_k}^{fw} + {v_k}^{bw} \leq 1$, for any k∈V

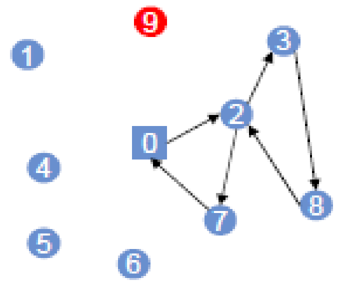
Decremental State-Space Relaxation (see Boland. 2006 & Righini. 2008)

Algorithm 2: **DecrementalSpaceSearch**()

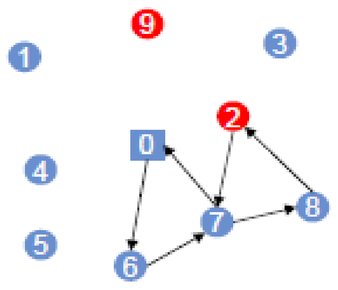
1. $N \leftarrow \emptyset, r^* \leftarrow \emptyset$
2. **While**(true)
3. Solve the pricing problem by the label-setting algorithm and obtain the optimal route r
4. **If** ($N_r = \emptyset$) //N_r is the set of customers visited more than once in r
5. $r^* \leftarrow r$
6. **Break**
7. **else**
8. $N \leftarrow N \cup N_r$
9. **Return** r^*



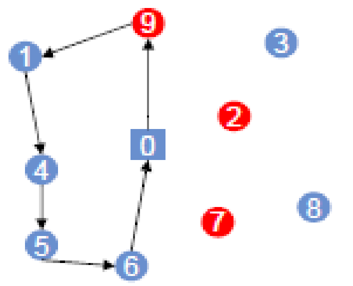
Step 1: $N = \{\}$



Step 2: $N = \{9\}$



Step 3: $N = \{2,9\}$



Step 4: $N = \{2,7,9\}$

Ng-path Relaxation

- for each customer i, define a neighborhood N_i called ng-set

$$N_i = \{j: the \ \triangle \ closest\ customers\ to\ i\} \cup \{i\}$$

- Let $V(L) = \{i_1, \dots, i_k\}$ be the customers visited in $p(L)$, then $\Pi(p)$ is a memory set of customers that L cannot be extended, which is given by

$$\Pi(p) = \{i_u \in V(L): i_u \in \bigcap_{s=u}^{k} N_{i_s}\}$$

A path $p = \{s, i_1, \dots, i_k\}$ satisfies this condition is called ng-path.

- When a new label L' is created, set $\Pi(L')$ is computed by

$$\Pi(L') = \Pi(L) \cap N_j \cup \{j\}$$

- In the dominance rule, $\mathcal{N}(L) \subseteq \mathcal{N}(L')$ is replaced by $\Pi(L) \subseteq \Pi(L')$

$N_1 = \{1,2\}, N_2 = \{2,1\}, N_3 = \{3,1\}$

$\Pi_0 = \{\}$

$\Pi_1 = \Pi_0 \cup N_1 = \{1\}$

$\Pi_2 = \Pi_1 \cap N_2 \cup \{2\} = \{1\} \cap \{2,1\} \cup \{2\} = \{1,2\}$

$\Pi_3 = \Pi_2 \cap N_3 \cup \{3\} = \{1,2\} \cap \{3,1\} \cup \{3\} = \{1,3\}$

$N_1 = \{1,2\}, N_2 = \{2,1\}, N_3 = \{3,2\}$

$\Pi_0 = \{\}$ $\Pi_1 = \{1\}$ $\Pi_2 = \{1,2\}$

$\Pi_3 = \Pi_2 \cap N_3 \cup \{3\} = \{1,2\} \cap \{3,2\} \cup \{3\} = \{2,3\}$

Completion Bound / Label Pruning

- $lb(L)$: a lower bound on the reduced cost of all feasible extensions in $\mathcal{E}(L)$ that reach the depot t
- If completion bound for $p(L) = c(L) + lb(L) > 0$, then the label L can be pruned without losing any negative reduced cost route

(Example) use a relaxed version of the labeling algorithm where only $c(L)$ and $q(L)$ are used. In this way, the dominance rules become much stronger and the algorithm can run much faster.

- Define $f(i,w) \coloneqq reduced\ cost\ of\ the\ path\ that\ is\ from\ vertex\ i\ to\ the\ depot\ t\ remaining\ capacity\ w$, then it can be calculated recursively by

$$f(i,w) = \begin{cases} 0, if\ i = 0 \\ \min\limits_{j \in \{k=1,\dots,n \mid k \neq i, q_k \leq w\}} f\left(j, w - q_j\right) + d_{ij}, if\ i \neq 0 \end{cases}$$

- $lb(L_i) = f(i, Q - q(L_i))$

Heuristic Pricing

except to prove the optimality of the current solution in the last CG iteration, fast and effective heuristics have been developed to find negative reduced cost variables

- Relaxing certain dominance rules
  - Consider only a restricted subset of customer resources
- Reducing the size of the network
  - Keep only the best incoming and outcoming arcs respect to the reduced cost
- Well-known heuristics
  - Tabu search (see Desaulniers .2008)

To ensure optimality, an exact algorithm must always be executed at least once, in the last CG iteration

THANK YOU FOR LISTENING

References

- Desrosiers (1995)_Time constrained routing and scheduling
- Feillet (2004)_ An exact algorithm for ESPPRC_ Application to some vehicle routing problems
- 【2005】《Column Generation》
- Boland (2006)_ Accelerated label setting algorithms for ERCSPP
- Righini (2006)_Symmetry helps：Bounded bi-directional dynamic programming for ESPPRC
- Righini (2008)_ New dynamic programming algorithms for ESPPRC
- Jepsen (2008)_Subset-Row Inequalities Applied to VRPTW
- Desaulniers (2008)_Tabu search, partial elementarity, and generalized k-path inequalities for VRPTW