KATHOLIEKE UNIVERSITEIT
LEUVEN

Combinatorial Optimization and Local Search Techniques

Solution homework 7

Results previous homework

Name	2500	5000	10000	20000	2500	5000	10000	20000	Avg	Correct
Erik	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,000	8
Yizheng	0,01	0,01	0,12	4,96	0,01	0,03	0,03	0,34	0,689	8
Inès	0,03	0,07	0,12	4,82	0,02	0,15	0,11	0,86	0,773	8
Qinyan	0,00	0,01	0,03		0,00	0,01	0,03		0,013	6
Ari	0,01	0,03	0,06		0,01	0,02	0,06	0,43	0,089	6
Omar	1,39	49,67	0,72		0,05	0,08	8,31		10,037	6

Results current homework

Name	2500	5000	10000	20000	2500	5000	10000	20000	Avg	Correct
Erik	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,000	8
Yizheng2	0,00	0,00	0,03	0,04	0,00	0,01	0,01	0,02	0,014	8
Inès	0,00	0,01	0,00	0,07	0,00	0,01	0,00	0,07	0,020	8
Inès fract	0,00	0,06	0,00	0,13	0,02	0,07	0,00	0,05	0,041	8
Inès fill	0,00	0,01	0,02	0,29	0,01	0,02	0,03	0,12	0,063	8
Omar alt	0,02	0,06	0,18	2,12	0,02	0,10	0,17	0,74	0,426	8
Omar fract	0,16	0,64	7,51	7,48	0,22	0,65	1,11	17,85	4,453	8
Qinyan fract	0,00	0,00	0,00		0,00	0,00	0,00		0,000	6
Qinyan fill	0,00	0,00	0,01		0,00	0,00	0,01		0,003	6
Ari	0,01	0,02	0,07		0,01	0,02	0,07		0,033	5
Yizheng1	0,01	0,01	0,17	0,75	0,01	0,03	0,11	0,40	0,186	3

Alternative branching strategy

```
#include “HW6.cpp"

mat del, upper, pointers, delay, remain, lower;
int lowerbound, nrstack, stack[100];

void init (void)
{    for (int i = 0; i < n; i++) x[i] = 0; del[0] = n; return;}

...

void procedure (void)
{    init ();
     sort ();
     start ();
     return;
}
```

Alternative branching strategy

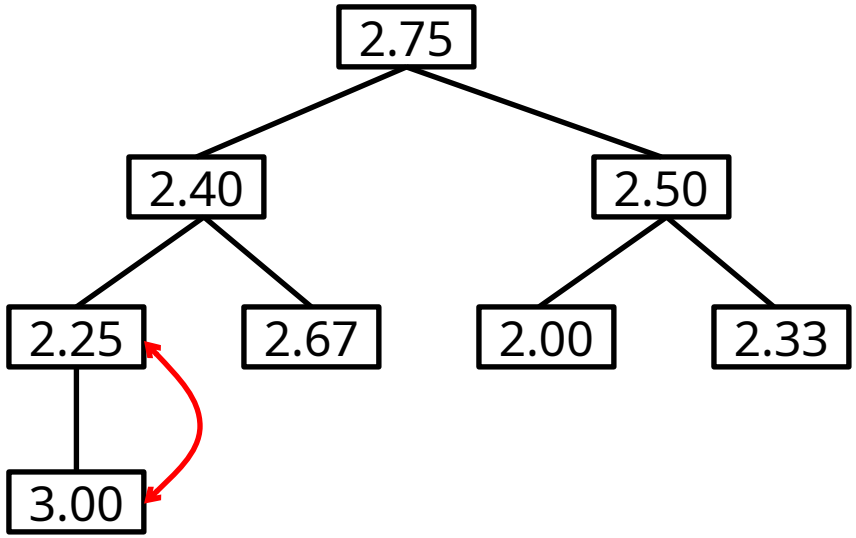
```
mat nr, xx, value, weight, minweight;
double f[maxn+1];

void siftup (int i, int n)
{    int j, notfinished, temp; double rootkey;
     rootkey=f[i]; temp = nr[i]; j=2*i; notfinished=(j<=n);
     while (notfinished)
     {          if (j<n && f[j+1]>f[j]) j++;
                if (f[j]<=rootkey) notfinished=0;
                else { f[i]=f[j]; nr[i]=nr[j]; i=j; j*=2; notfinished=(j<=n); }
     }
     f[i]=rootkey; nr[i]=temp; return;
}
```

Alternative branching strategy

```
void sort (void)
{    int i;
     for (i = 0; ++i <= n;) {f[i] = (double) c[i]/a[i]; nr[i] = i;}
     for (i=n/2; i>1; i--) siftup (i, n);
     for (i=n; i>1; i--)
     {              siftup (1, i);
                    xx[i-1]=nr[1]; value[i-1]=c[nr[1]]; weight[i-1]=a[nr[1]];
                    f[1]=f[i]; nr[1]=nr[i];
     }
     xx[0]=nr[1]; value[0]=c[nr[1]]; weight[0]=a[nr[1]];
     minweight[0] = weight[0]; for (i = 0; ++i < n;)
                    if (weight[i] < minweight[i-1]) minweight[i] = weight[i];
                    else minweight[i] = minweight[i-1];
     return;
}
```

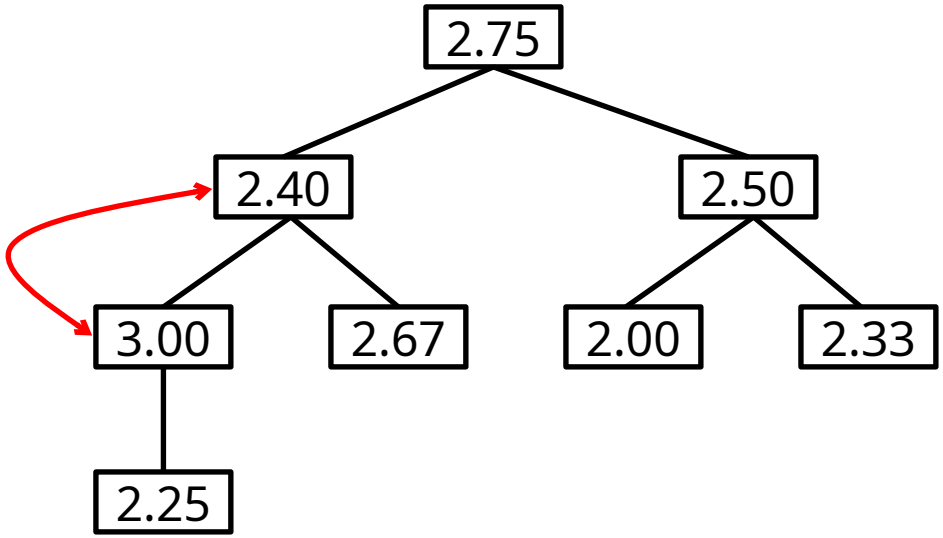
Alternative branching strategy

i	a	c	f	nr
0				
1	4	11	2.75	1
2	5	12	2.40	2
3	2	5	2.50	3
4	8	18	2.25	4
5	3	8	2.67	5
6	1	2	2.00	6
7	6	14	2.33	7
8	7	21	3.00	8



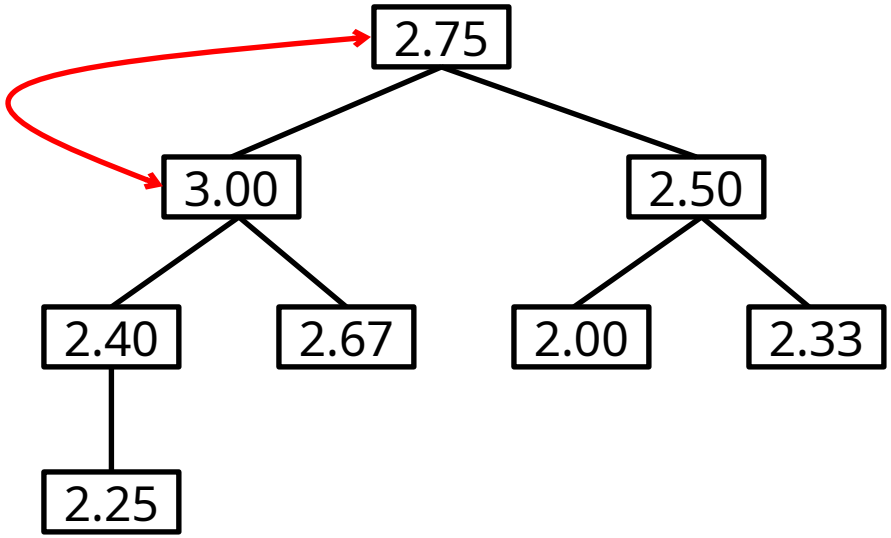
Alternative branching strategy

i	a	c	f	nr
0				
1	4	11	2.75	1
2	5	12	2.40	2
3	2	5	2.50	3
4	7	21	3.00	8
5	3	8	2.67	5
6	1	2	2.00	6
7	6	14	2.33	7
8	8	18	2.25	4



Alternative branching strategy

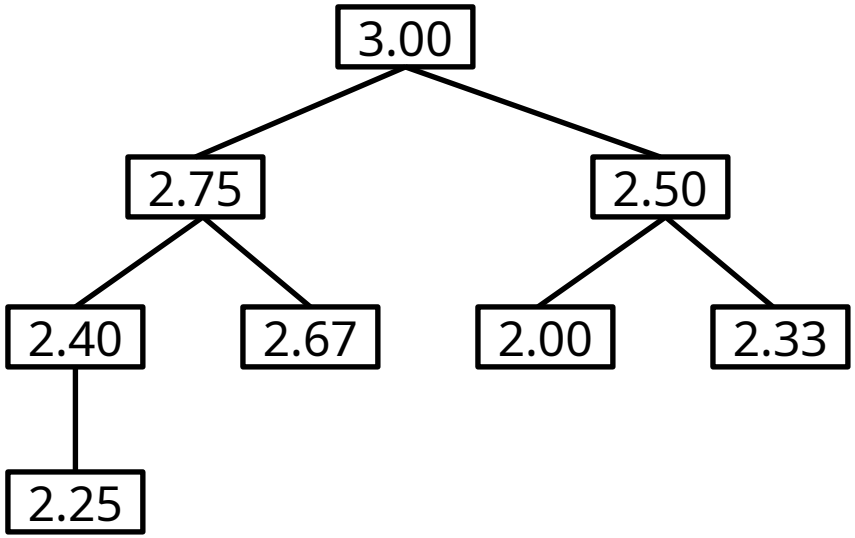
i	a	c	f	nr
0				
1	4	11	2.75	1
2	7	21	3.00	8
3	2	5	2.50	3
4	5	12	2.40	2
5	3	8	2.67	5
6	1	2	2.00	6
7	6	14	2.33	7
8	8	18	2.25	4

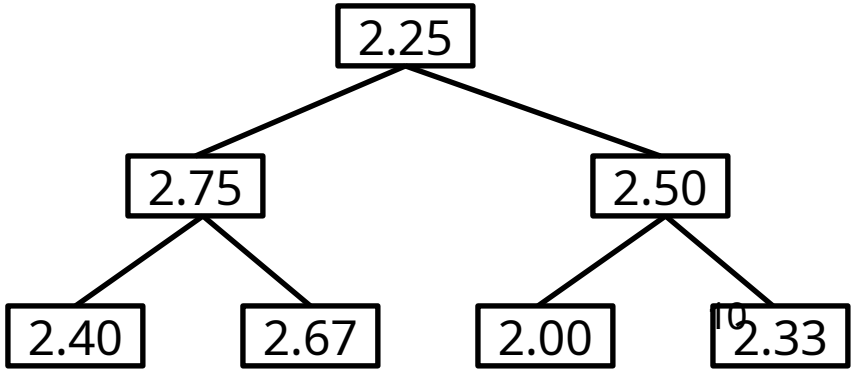


Alternative branching strategy

i	a	c	f	nr
0				
1	7	21	3.00	8
2	4	11	2.75	1
3	2	5	2.50	3
4	5	12	2.40	2
5	3	8	2.67	5
6	1	2	2.00	6
7	6	14	2.33	7
8	8	18	2.25	4

i	xx	value	weight
7	8	21	7





Alternative branching strategy

```
void start (void)
{    int i, over, left, fract, ub, lb;
     left = b; lowerbound = 0; for (i = n; --i >= 0;)
     {           if (weight[i] <= left) {x[xx[i]] = 1; left -= weight[i]; lowerbound += value[i];}
                 else {ub = lowerbound + left*value[i]/weight[i]; fract = i; break;}
     }
     if (i < 0) ub = lowerbound;
     else
     {           over = left; lb = lowerbound;
                 while (--i >= 0 && over >= minweight[i]) if (weight[i] <= over)
                 {           x[xx[i]] = 1; over -= weight[i]; lowerbound += value[i];}
     }
     if (ub > lowerbound) {pointers[1] = -1; recursion (1, fract, left, lb);}
     return;
}
```

Alternative branching strategy

```
void recursion (int lev, int fract, int left, int lb)
{    int i, i1, over, low, ub, point;
     point = pointers[lev]; for (i = fract; ++i < del[lev-1];)
     {           low = lb - value[i]; over = left + weight[i];
                 for (i1 = fract; i1 >= 0; i1--)
                 {           if (weight[i1] <= over) {over -= weight[i1]; low += value[i1];}
                             else {ub = low + over*value[i1]/weight[i1]; break;}
                 }
                 if (i1 < 0) ub = low;
                 upper[++point] = ub; delay[point] = i1; remain[point] = over; lower[point]
     = low;
                 nrstack = 0; while (--i1 >= 0 && over >= minweight[i1]) if (weight[i1] <=
     over)
                 {           stack[nrstack++] = i1; over -= weight[i1]; low += value[i1];}
```

Alternative branching strategy

```
        if (low > lowerbound)
        {       for (i1 = delay[point]; ++i1 < n;) x[xx[i1]] = 1;
                for (i1 = delay[point]; i1 >= 0; i1--) x[xx[i1]] = 0;
                while (--nrstack >= 0) x[xx[stack[nrstack]]] = 1;
                for (i1 = lev; --i1 > 0;) x[xx[del[i1]]] = 0;
                x[xx[i]] = 0; lowerbound = low;
        }
}
```

Alternative branching strategy

```
    pointers[lev+1] = point;
    over = left; low = lb; for (i1 = fract; --i1 >= 0;)
    {           if (over >= weight[i1]) {over -= weight[i1]; low += value[i1];}
                else {ub = low + over*value[i1]/weight[i1]; break;}
    }
    if (i1 < 0) ub = low;
    if (ub > lowerbound)
    {           del[lev] = fract; recursion (lev+1, i1, over, low);}
    for (i = fract, point = pointers[lev]; ++i < del[lev-1];)
    {           if (upper[++point] <= lowerbound) continue;
                del[lev] = i; recursion (lev+1, delay[point], remain[point], lower[point]);
    }
    return;
}
```