User Scheduling in 5G

Ziyad Benomar,
Redallah Mouhoub

February 2020

Content

- Formulation of the ILP problem
- Preprocesing
- Greedy algorithm
- Dynamic Programming
- Alternative Dynamic Programming
- Branch and Bound algorithm
- Stochastic Online algorithm
- Appendix (Implementation)

Formulation of the ILP problem

ILP problem

$$\text{Maximise} \quad \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{m\in\mathcal{M}} r_{k,m,n}x_{k,m,n}$$

$$\text{subject to} \quad \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{m\in\mathcal{M}} p_{k,m,n}x_{k,m,n} \leq p$$

$$\sum_{k\in\mathcal{K}}\sum_{m\in\mathcal{M}} x_{m,k,n} = 1 \quad \forall n \in \mathcal{N}$$

$$x_{k,m,n} \in \{0,1\} \quad \forall k \in \mathcal{K}, n \in \mathcal{N}, m \in \mathcal{M}$$

Preprocessing

- Remove triplets (k, m, n) that, if chosen, obviously prevent any solution to be feasible.
- Remove IP-dominated terms of an instance of the IP problem.
- Remove LP-dominated terms of an instance of the LP problem.

Preprocessing

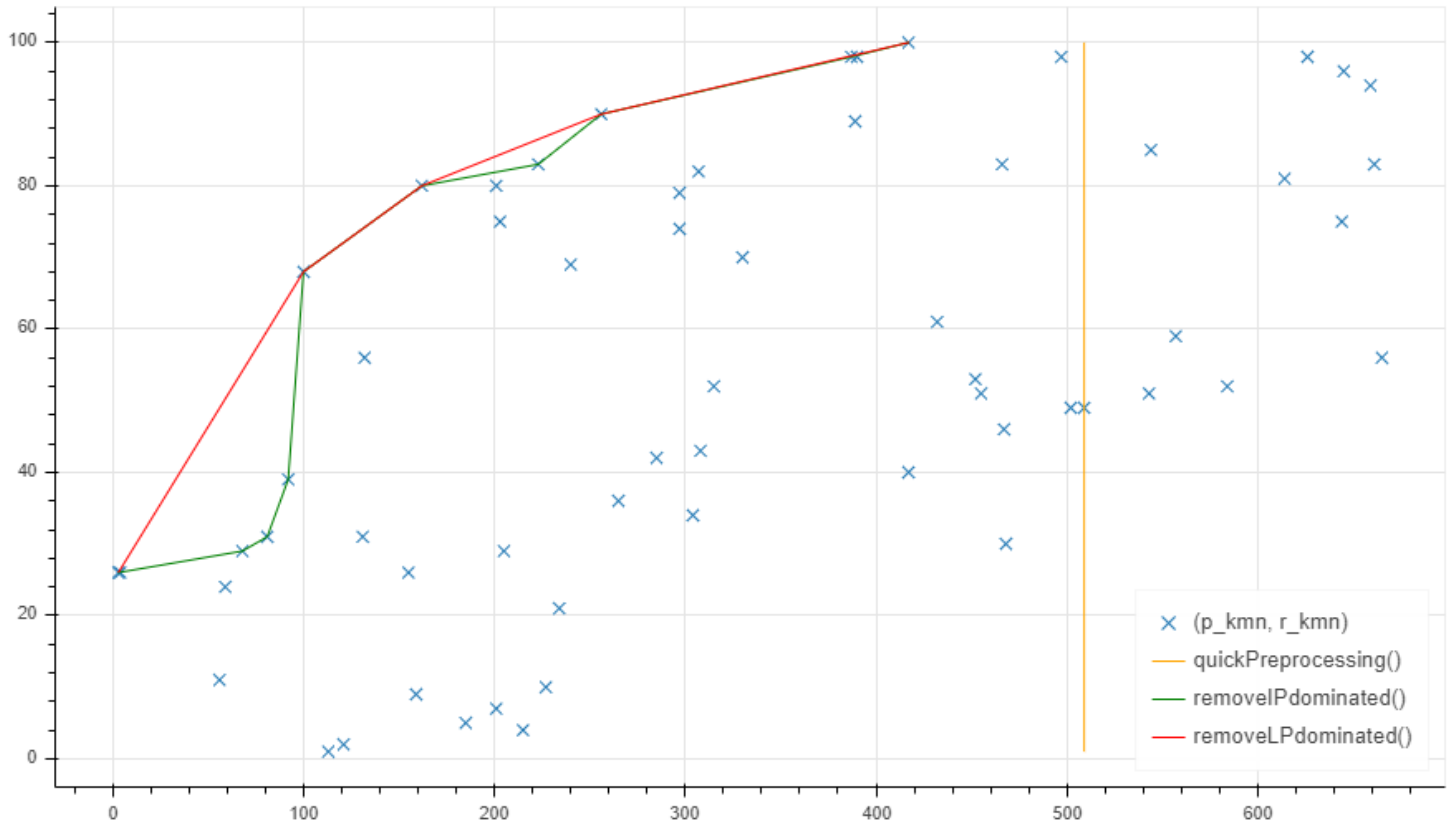


Figure: "test5.txt", n = 9

Preprocessing

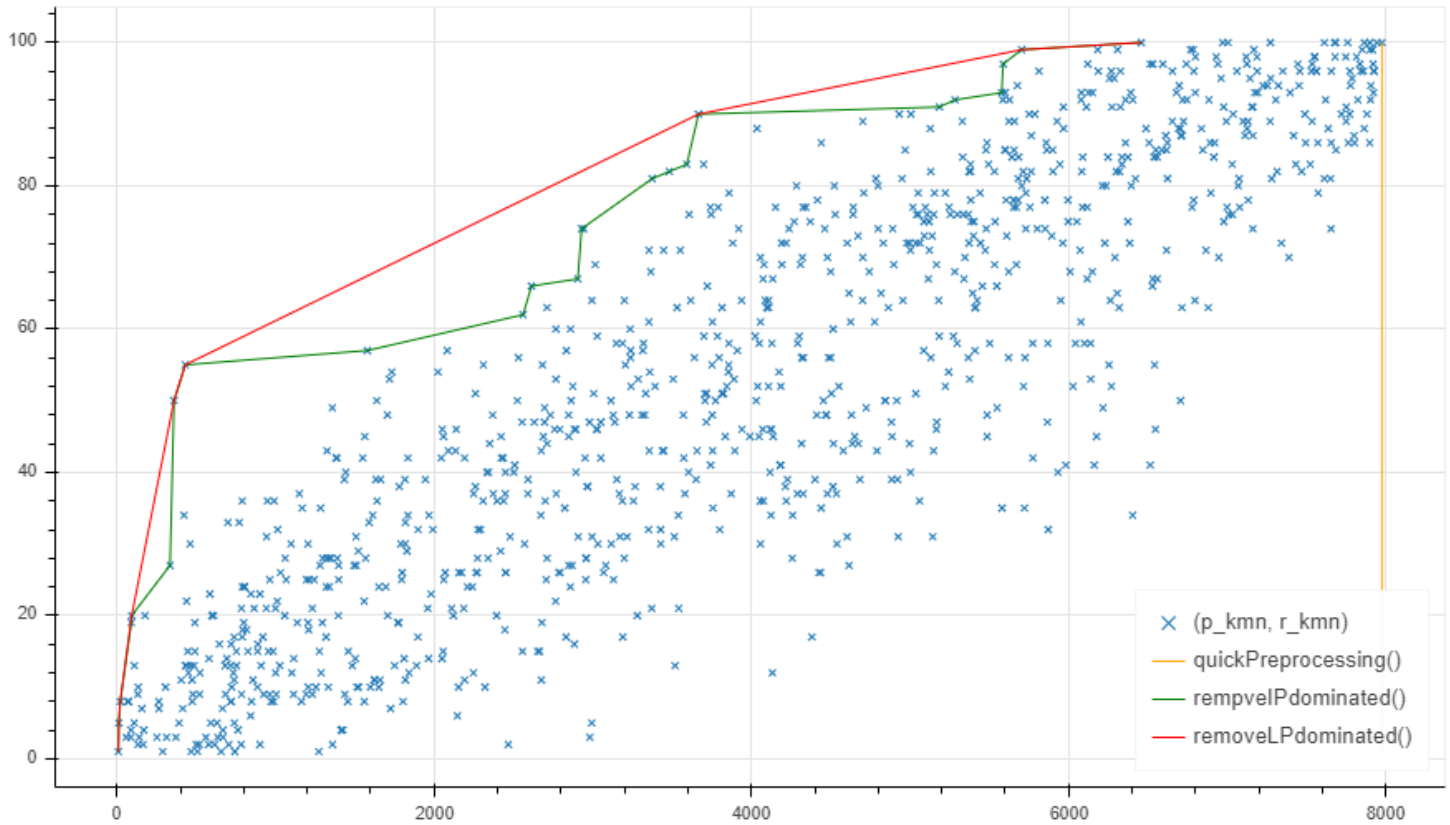


Figure: "test4.txt", n = 36

Preprocessing

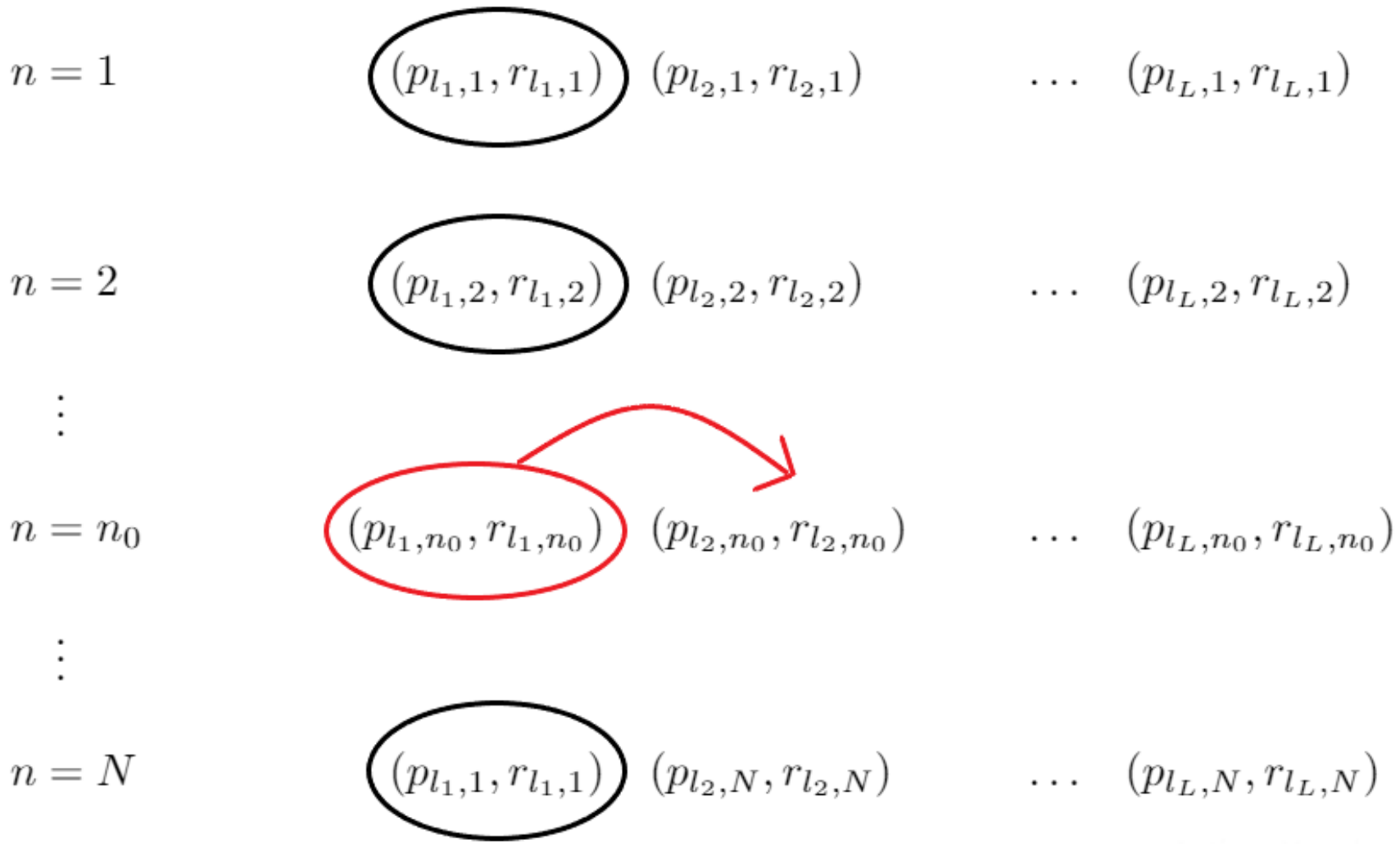
results

Complexity

The time complexity of the preprocessing is $\mathcal{O}(NKM \log(KM))$

	"test1.txt"	"test2.txt"	"test3.txt"	"test4.txt"	test5.txt"
initially	24	24	24	614400	2400
quickPreprocessing()	24	0	24	614400	1954
removeIPdominated()	10	0	13	14687	300
removeLPdominated()	8	0	9	4974	179

Greedy algorithm



Complexity

The time complexity of the function LP.solve() is $\mathcal{O}(NKM \log(NKM))$ and its space complexity is $\mathcal{O}(NKM)$

$solveLP()$	"test1.txt"	"test3.txt"	"test4.txt"	test5.txt"
Budget power	100	100	16000	1000
Used power	78	100	16000	1000
Data rate	365	372.15384	9870.322	1637
Run time (ms)	12	5	13	326
IP-solution	true	false	false	true

First approach

$\forall\, 0 \leq n < N,\ \forall\, 0 < p \leq P$

$$R(n,p) = max\{R(n+1, p - p_{k,m,n}) + r_{k,m,n} \mid (k,m) \in \mathcal{K} \times \mathcal{M}\}$$
$$R(N,p) = R(n,0) = 0$$

Second approach

$\forall\, 0 \leq n < N,\ \forall\, 0 \leq r < U$

$$P(n,r) = min\{P(n+1, r - r_{k,m,n}) + p_{k,m,n} \mid (k,m) \in \mathcal{K} \times \mathcal{M}\}$$
$$P(n,0) = 0;\ P(N-1,r) = p_{k,m,n} \text{ if } r_{k,m,n} = r, \text{ and } +\infty \text{ otherwise}$$

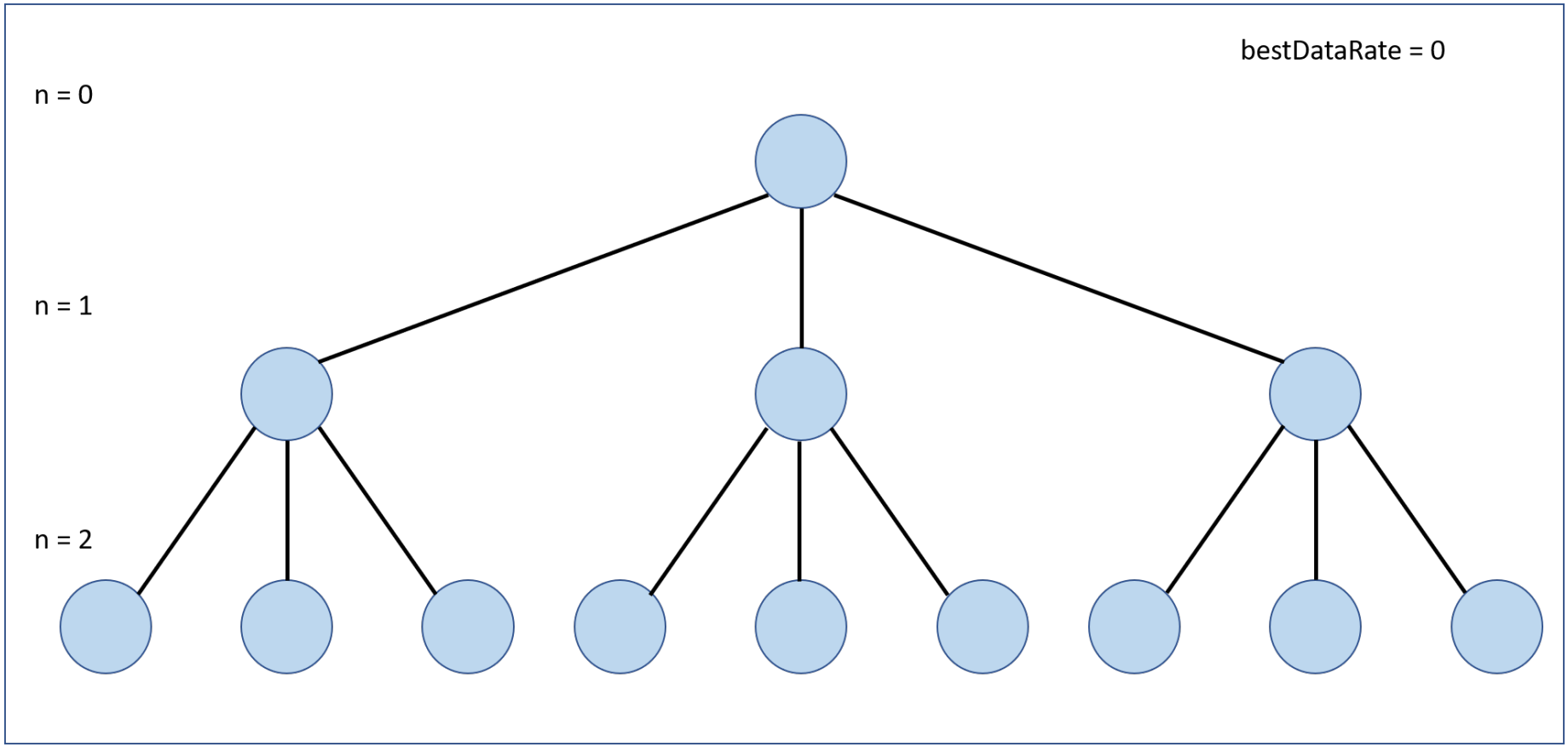
Complexity

The time complexity of DP.solve() is $\mathcal{O}(PKMN)$
The time complexity of DP.solveAlternative() is $\mathcal{O}(KMN(U + \log(KMN)))$

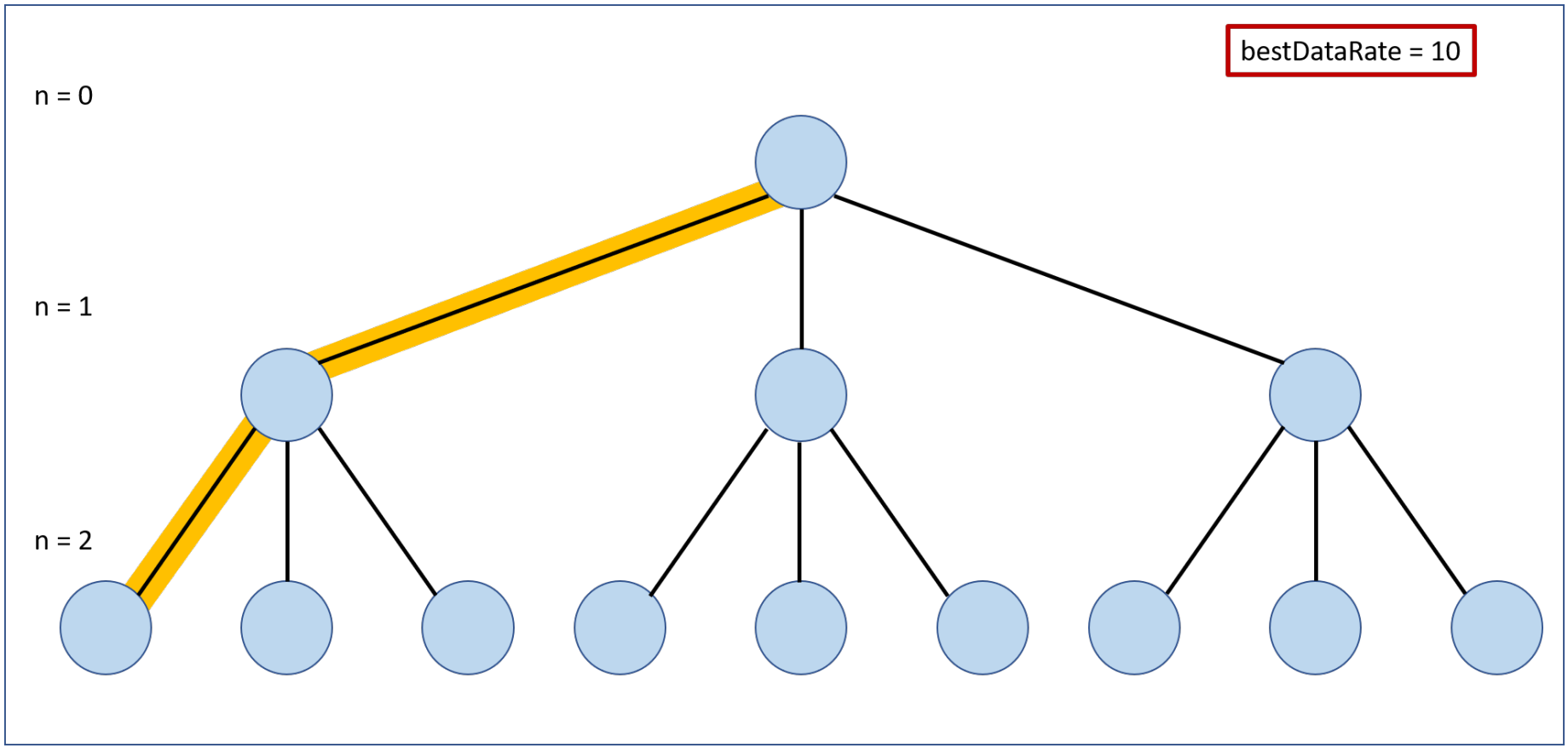
solveDP()	"test1.txt"	"test3.txt"	"test4.txt"	test5.txt"
Budget power	100	100	16000	1000
Used power	78	68	16000	1000
Data rate	365	350	9870	1637
Run time (ms)	16.1	0.2	8978.0	18.0

solveDPalt()	"test1.txt"	"test3.txt"	"test4.txt"	test5.txt"
Budget power	100	100	16000	1000
Used power	78	68	15999	1000
Data rate	365	350	9870	1637
Run time (ms)	10.0	0.0	4173.0	20.0

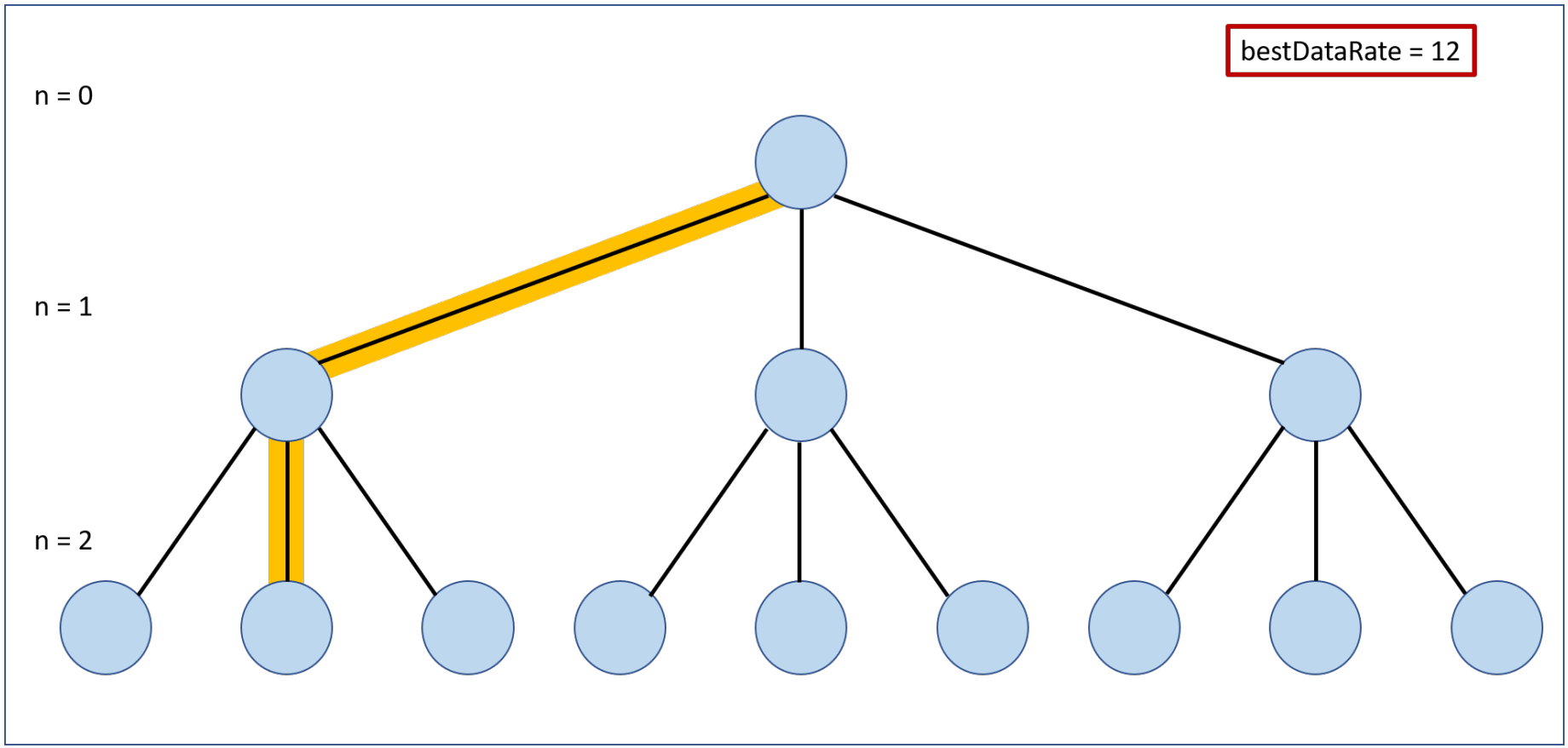
Branch-and-Bound algorithm



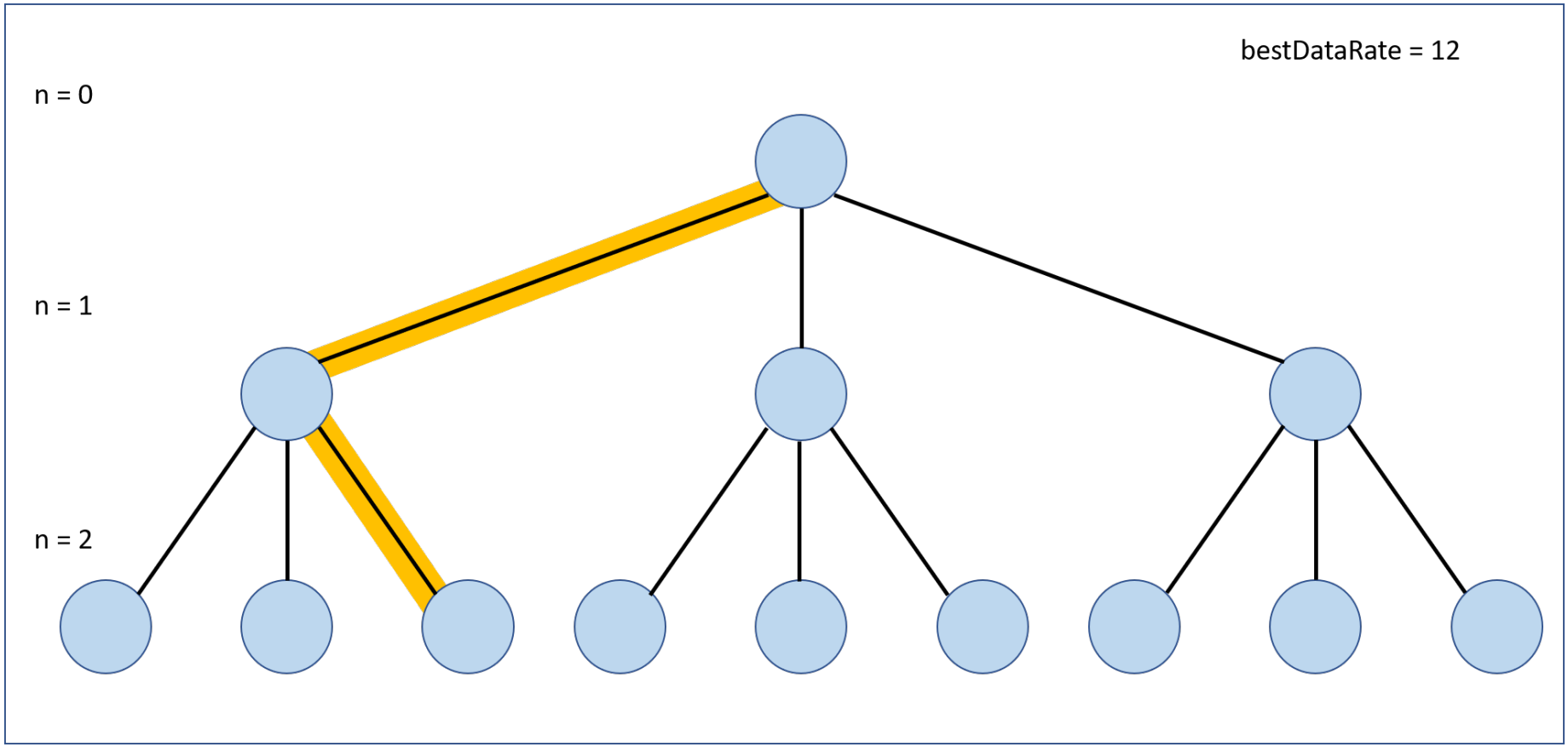
Branch-and-Bound algorithm



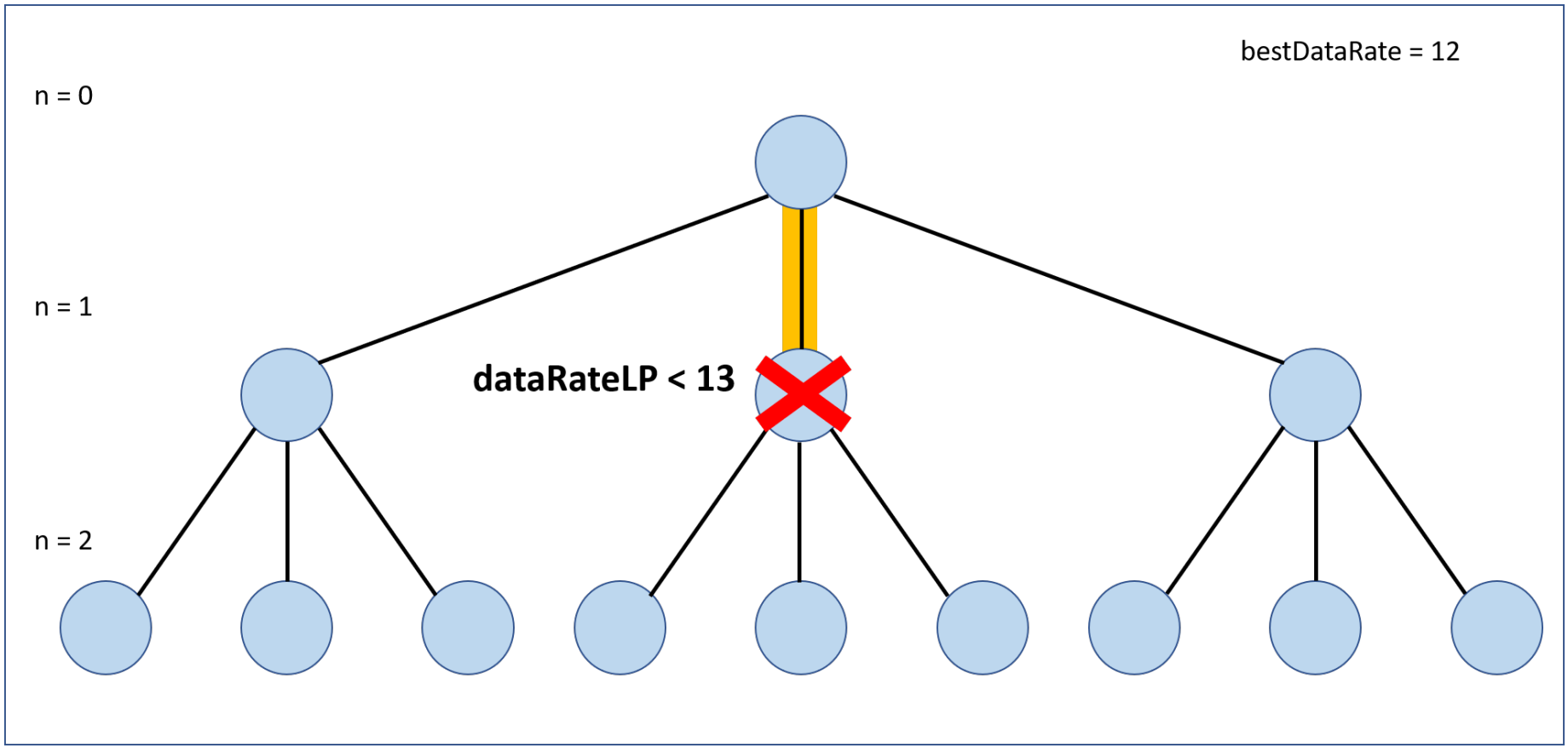
Branch-and-Bound algorithm



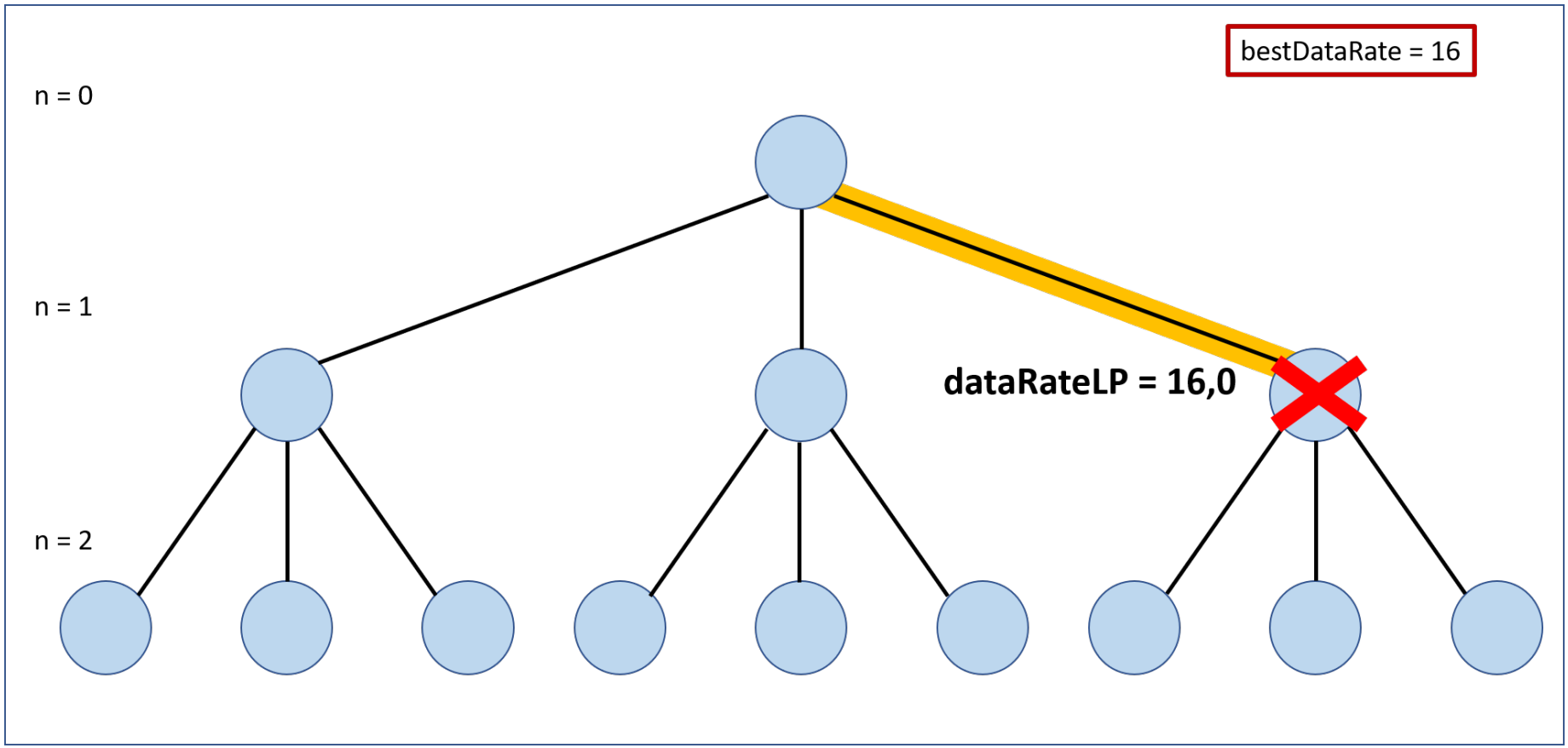
Branch-and-Bound algorithm



Branch-and-Bound algorithm



Branch-and-Bound algorithm



Branch-and-Bound algorithm

results

- Initialization of bestDataRate ?

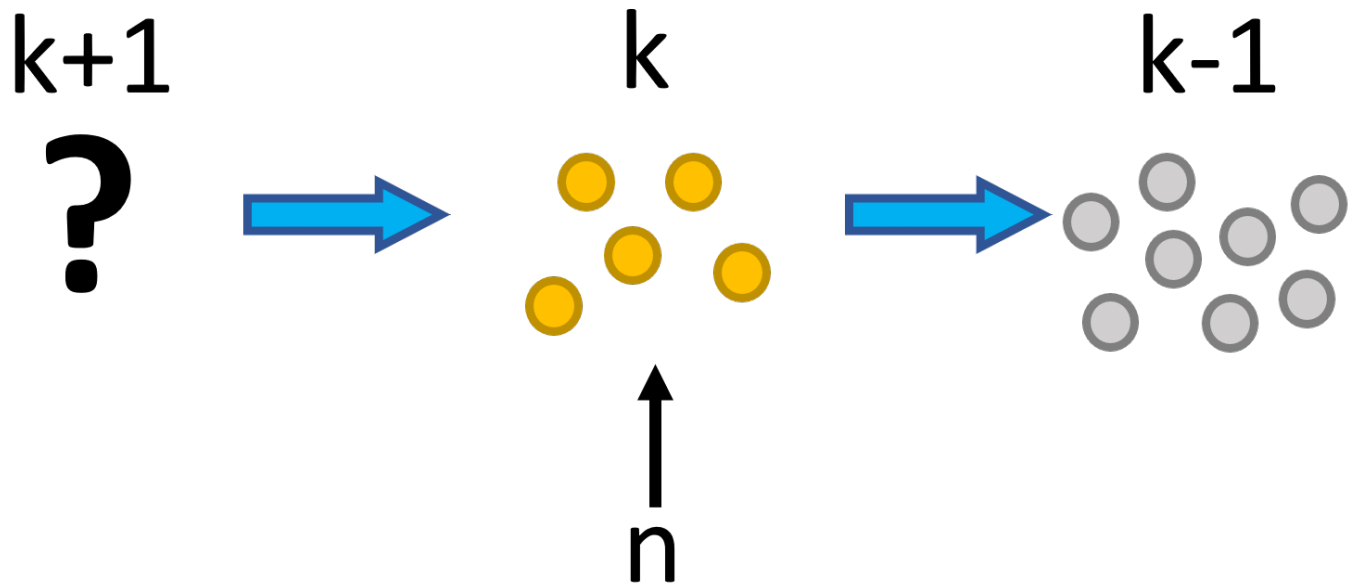
Complexity

The time complexity of the function BB.solve() is $\mathcal{O}((KM)^N)$

$solveBB()$	"test1.txt"	"test3.txt"	"test4.txt"	test5.txt"
Budget power	100	100	16000	1000
Used power	78	68	16000	1000
Data rate	365	350	9870	1637
Run time (ms)	22.0	1.2	949.0	1.0
Explored nodes	1	10	4536	1

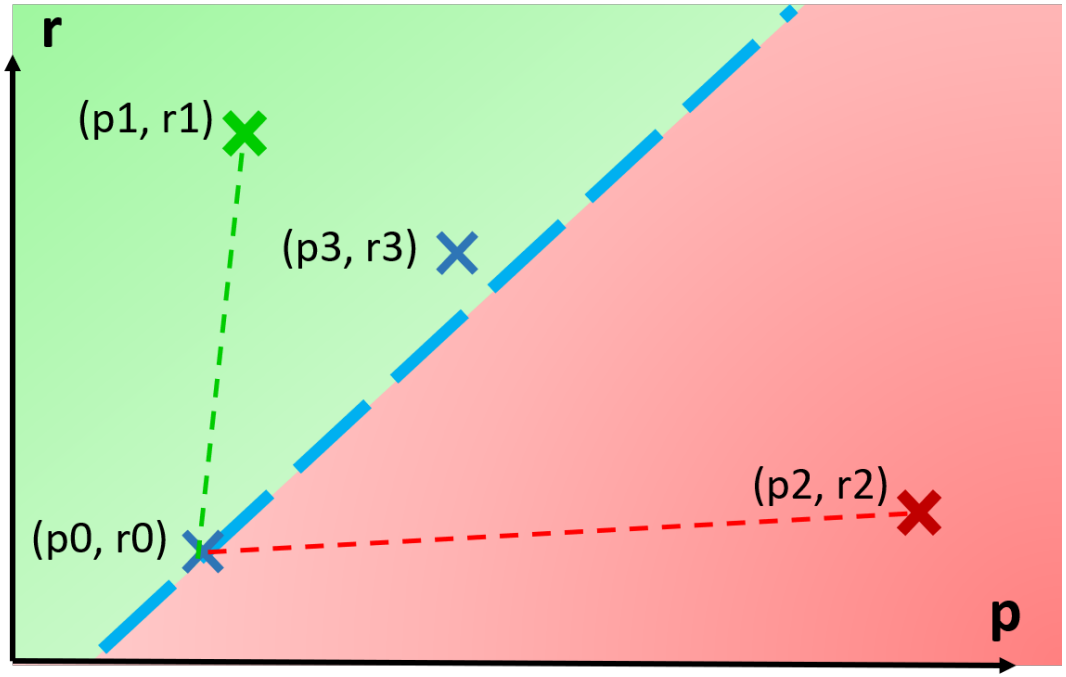
Online programming

- Choose the best point among the current ones
- Decide if it's good enough to be picked or not



Online programming

Comparing points

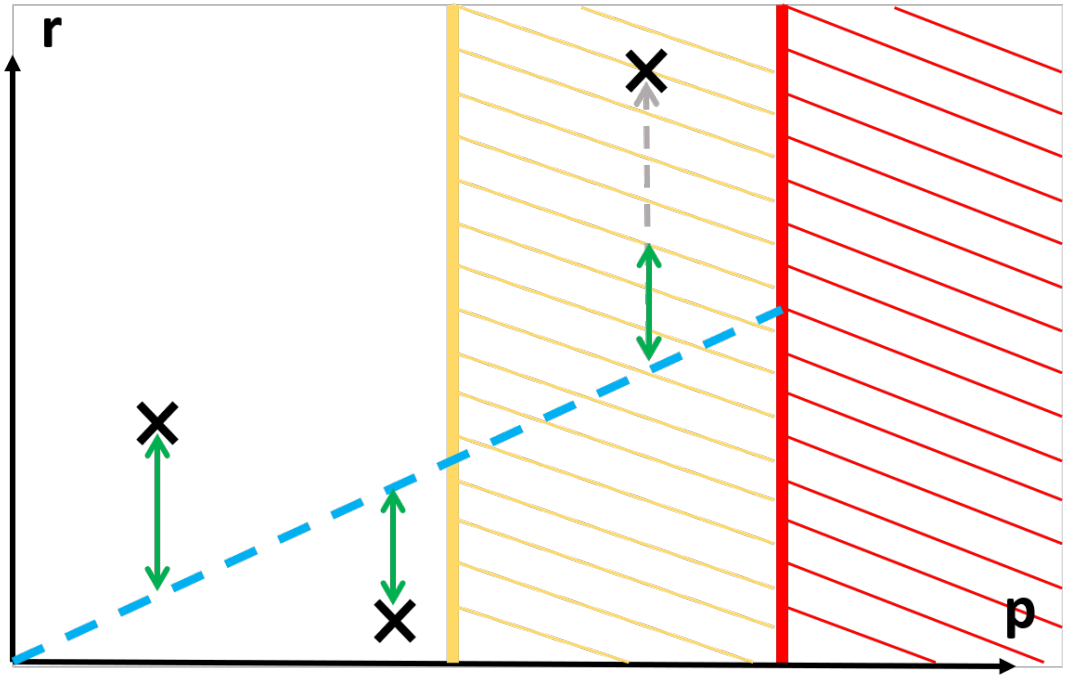


$$(p_2, r_2) \text{ is better than } (p_1, r_1) \iff \frac{r_2 - r_1}{p_2 - p_1} \geq e^* = \frac{1}{p^m r^m} \sum_{p=1}^{p^m} \sum_{r=1}^{r^m} \frac{r}{p}$$

$$\iff r_2 - e^* p_2 \geq r_1 - e^* p_1$$

Online programming

the quality function



- limitAllowedBudget = p - $(|\mathcal{N}_k|\text{-}1)(1+\frac{p^{max}}{M})$
- If $p_{k,m} > \frac{p}{|\mathcal{N}_k|}$: $quality(p_{k,m}, r_{k,m}) = (r_{k,m} - e^* p_{k,m})\frac{p}{p_{k,m}|\mathcal{N}_k|}$

Online programming

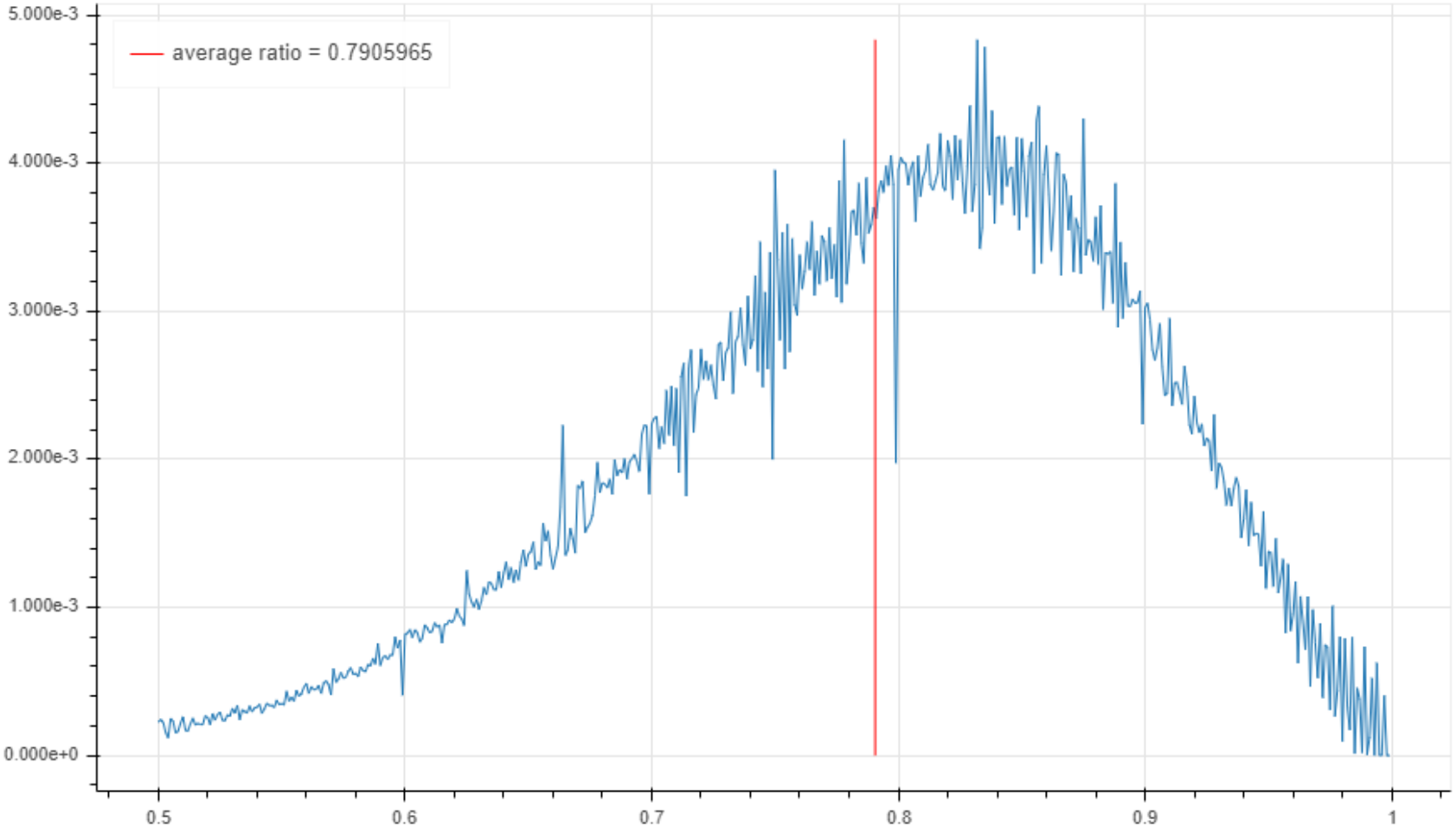
is the point good enough ?

- Proportion of points that are not better

Online programming

results over 500000 experiences

- Competitive ratio : 0.7905965
- 0.00273 % of the solutions are not complete
- Run time : 5 minutes 2 seconds



Appendix

class Couple

int k,m;

class Doubly

int k,m,n;
boolean IpDominated;
Doubly next,prev;

class Instance

```
int N,K,M,P;
int[ ][ ][ ] p,r;
Doubly[ ] t;
public Instance(String path);
```

```
<terminated> Instance [Java Application] C:\Program Files\Java\jdk-12.0.1\bin\javaw.exe (9 févr. 2020 à 22:28:55)
K = 3
M = 2
N = 4
P = 100
[HEAD] --> t[0] : (1,0,0) --> (0,0,0) --> (2,0,0) --> (0,1,0) --> (1,1,0) --> (2,1,0) --> [TAIL] .
[HEAD] --> t[1] : (1,0,1) --> (0,0,1) --> (1,1,1) --> (2,0,1) --> (0,1,1) --> (2,1,1) --> [TAIL] .
[HEAD] --> t[2] : (2,0,2) --> (2,1,2) --> (0,0,2) --> (1,0,2) --> (1,1,2) --> (0,1,2) --> [TAIL] .
[HEAD] --> t[3] : (0,0,3) --> (2,0,3) --> (1,0,3) --> (1,1,3) --> (2,1,3) --> (0,1,3) --> [TAIL] .
```

Figure: Instance("test3.txt")

class Solution

Instance ins;
Couple[ ] x;
int k, m, n;
float lambda;

$$\forall n' \neq n : x_{k',m',n'} = 1 \text{ with } (k', m') = x[n'],$$
$$x_{k,m,n} = \lambda,\ x_{k_n,m_n,n} = 1 - \lambda \text{ with } (k_n, m_n) = x[n].$$

```
{(k,m,n) | x_kmn=1 } = { (0,1,0), (2,1,2), (1,1,3) }
For n = 1 : x_{0,1,1} = 0.82051283, x_{0,0,1} = 0.17948717
The initial budget power : 100
The total used power     : 100.0
The total data rate      : 372.15384
```

The solution for the LP problem "test3.txt"

class Preprocessing

- public static boolean preprocess(Instance ins)
- public static boolean quickPreprocessing(Instance ins)
- public static int removeIPdominated(Instance ins)
- public static int findLPdominated(Instance ins)
- public static Doubly[ ] removeLPdominated(Instance ins)

Implementation

Greedy algorithm

class LP

- public static boolean solveSubLP(Solution sol, Doubly sortedE, int n0, int leftBudgetPower)
- public static Solution solve(Instance ins)

class DP

- public static int[ ][ ] computeR(Instance ins)
- public static Solution solve(Instance ins)
- public static int[ ][ ] computeP(Instance ins, int U)
- public static Solution solveAlternative(Instance ins)

Implementation

Branch-and-bound algorithm

class BB

- Solution bestSolution;
- int bestDataRate;
- Doubly sortedE;
- public boolean solveRecBB(Solution sol, int n0, int leftBudgetPower)
- public static Solution solve(Instance ins)

The End