Patient-Nurse-Room Assignment

Contents

1 Problem Description

In this project, you are required to solve a healthcare scheduling problem involving the assignment of patients to rooms and nurses to rooms in a hospital. The goal is to develop an optimization model using Mixed Integer Programming (MIP) that minimizes the total delay in patient admissions while adhering to a set of operational and resource constraints. This problem is inspired by the *Integrated Healthcare Timetabling Competition 2024* (https://ihtc2024.github.io/).

1.1 Overview

The hospital operates with a set of patients, rooms, and nurses, each with specific characteristics:

- **Patients** are defined by their release date, indicating the earliest day they can be admitted, and a due date, specifying the latest allowable admission day. They also have a required length of stay.
- **Rooms** are characterized by fixed capacities that determine the maximum number of patients they can accommodate at any given time. Additionally, some patients may have incompatibilities with certain rooms, restricting their assignment.
- **Nurses** have predefined rosters detailing their availability across days. Each occupied room must have one (and only one) assigned nurse on every day of the scheduling horizon. Nurses can be assigned to a maximum of 3 rooms in a given day.

The problem requires assigning each patient to a room and ensuring that every occupied room has an assigned nurse during each day. These assignments must satisfy several constraints: no room may exceed its capacity, nurse assignments must respect their availability, and patients must only be assigned to compatible rooms. The primary objective is to minimize the total delay in admissions, defined as the sum of the differences between actual admission dates and the release dates of all patients.

1.2 Constraints

1. A patient's length of stay must be respected, meaning they occupy a room for the full duration of their stay.
2. A room cannot exceed its capacity on any given day.
3. Patients can only be assigned to rooms that they are compatible with.
4. Each occupied room must have exactly one assigned nurse per day.

5. Nurses can only be assigned to rooms on days when they are available.
6. Nurses can be assigned to a maximum of 3 different rooms in the same day.
7. Admission days must be within the specified release and due dates.

1.3 Decisions

- **Patient-to-Room Assignment**: Assign each patient to a single room, ensuring that the same room is used for the entire duration of the patient's stay.
- **Admission Day**: Determine the admission day for each patient within the allowed range of their release and due dates.
- **Nurse-to-Room Assignment**: Assign one nurse to each occupied room on every day of the scheduling period.

1.3.1 Objective

The objective is to minimize the **total admission delay**, defined as the sum of the differences between the admission day and the release date of patients.

1.4 Input Data

1.4.1 Days

- Number of days: 7

1.4.2 Rooms

Room ID	Capacity
0	3
1	2
2	2
3	2
4	2

Table 1: Rooms and their capacities.

1.4.3 Patients

Patient ID	Length of Stay	Release Date	Due Date	Incompatible Room IDs
0	3	0	3	[]
1	2	0	4	[1]
2	4	0	5	[2]
3	3	0	4	[0]
4	2	1	6	[]
5	3	1	6	[3]
6	1	0	2	[]
7	2	2	5	[1]
8	3	0	6	[]
9	3	2	7	[]
10	2	3	6	[4]
11	2	4	7	[]
12	1	4	7	[3]
13	2	1	5	[]
14	3	2	6	[]
15	1	3	4	[]
16	2	0	3	[2]
17	3	4	7	[]
18	2	5	7	[]
19	3	0	6	[]

Table 2: List of patients with their stay requirements and room incompatibilities.

1.4.4 Nurses

Nurse ID	Working Shifts (Days)
0	{0, 1, 2, 3}
1	{1, 2, 3, 4}
2	{0, 1, 3, 4}
3	{2, 3, 5, 6}
4	{4, 5, 6}

Table 3: Nurses and the days they are available to work.

IDEA: La scelta di formulare il problema come un *Mixed-Integer Program* (MIP) nasce dalla presenza di variabili binarie (ad esempio, le scelte di ammissione dei pazienti e di assegnazione degli infermieri) e da vincoli di natura combinatoria (come il rispetto della capacità delle stanze o la compatibilità dei pazienti con le sale). In particolare, la necessità di gestire sia vincoli lineari sia scelte discrezionali (ammettere o meno un paziente in un determinato giorno, assegnare o meno un'infermiera a una stanza) rende la programmazione lineare intera la metodologia più adatta a garantire soluzioni esatte e comprovate in termini di ottimalità.

L'algoritmo di risoluzione tipico per i MIP è il *Branch-and-Bound*, eventualmente esteso con tecniche di *Branch-and-Cut*. L'idea di base consiste nell'esplorare in maniera

sistematica l'insieme di tutte le possibili soluzioni (ramificazione, o *branching*), dividendole in sottoproblemi. A ciascun nodo dell'albero di esplorazione si risolve una versione rilassata del problema (ad esempio, consentendo variabili continue al posto di quelle intere) per ottenere un limite inferiore. Se la soluzione rilassata non è ammissibile (o se il suo valore è peggiore di un limite superiore già noto), il sottoalbero corrispondente viene scartato (o *bound*). In aggiunta, si possono impiegare *tagli* (cutting planes) che sfruttano proprietà poliedrali del problema per restringere ulteriormente lo spazio delle soluzioni e accelerare la ricerca dell'ottimo. Questo approccio consente di pervenire ad una soluzione garantita come *ottimale* una volta che tutti i rami rilevanti dell'albero siano stati esplorati o opportunamente potati.

2 Q1: Modeling Choices

Write here your modeling decisions

We address a patient–nurse–room assignment problem with the following sets:

- I: set of patients, indexed by i.
- R: set of rooms, indexed by r.
- D: set of days, indexed by t.
- N: set of nurses, indexed by n.
- $Incomp_i \subseteq R$: rooms incompatible with patient i.
- $Shifts_n \subseteq D$: days on which nurse n is available.

Parameters:

- Cap_r: capacity of room r.
- L_i: length of stay required by patient i.
- Rel_i: earliest admission day (release date) for patient i.
- Due_i: latest admission day (due date) for patient i.

Decision Variables:

$$x_{i,r,d} = \begin{cases} 1, & \text{if patient } i \text{ starts in room } r \text{ on day } d, \\ 0, & \text{otherwise,} \end{cases}$$

$$y_{r,d,n} = \begin{cases} 1, & \text{if nurse } n \text{ is assigned to room } r \text{ on day } d, \\ 0, & \text{otherwise,} \end{cases}$$

$$z_{r,d} = \begin{cases} 1, & \text{if room } r \text{ is occupied on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

2.0.1 Objective Function

$$\min \sum_{i \in I} \sum_{r \in R} \sum_{d \in D \,:\, Rel_i \le d \le Due_i} (d - Rel_i)\, x_{i,r,d},$$

which minimizes the total admission delay: each patient's start day d is penalized by how much it exceeds Rel_i.

2.0.2 Constraints

(1) Unique Admission

$$\sum_{r \in R} \sum_{d \in D} x_{i,r,d} = 1 \quad \forall i \in I.$$

Interpretation: Ogni paziente i deve essere ammesso una volta sola: deve dunque scegliere esattamente una stanza r e un giorno di inizio d.

(2) Room Capacity

$$\sum_{\substack{i \in I,\, d \in D: \\ d \le t < d + L_i}} x_{i,r,d} \;\le\; Cap_r \quad \forall r \in R,\ \forall t \in D.$$

Interpretation: In qualsiasi giorno t, il numero di pazienti presenti in stanza r non deve superare la capacità Cap_r. Se un paziente i inizia il proprio soggiorno il giorno d, occupa la stanza nei giorni $d, d+1, \ldots, d+L_i-1$. Il vincolo somma quindi tutte le ammissioni i cui soggiorni includono il giorno t.

(3) Room Incompatibility

$$x_{i,r,d} = 0 \quad \forall i \in I,\ \forall r \in Incomp_i,\ \forall d \in D.$$

Interpretation: Se il paziente i è incompatibile con la stanza r, non può essere assegnato a tale stanza in nessun giorno.

(4) Admission Window

$$x_{i,r,d} = 0 \quad \text{se } d < Rel_i \;\text{ oppure }\; d > Due_i.$$

Interpretation: Il giorno di inizio d per il paziente i deve obbligatoriamente cadere nel suo intervallo di ammissione consentito, compreso tra la data di rilascio Rel_i e la data di scadenza Due_i.

(5) Exactly One Nurse in Each Occupied Room Per definire se una stanza r è occupata il giorno t, si introduce $z_{r,t}$. Questo variabile binaria collega la somma dei

pazienti presenti a $z_{r,t}$:

$$\sum_{\substack{i\in I,\, d\in D:\\ d\le t<d+L_i}} x_{i,r,d} \;\le\; Cap_r \cdot z_{r,t}, \qquad \sum_{\substack{i\in I,\, d\in D:\\ d\le t<d+L_i}} x_{i,r,d} \;\ge\; z_{r,t},$$

$$\sum_{n\in N} y_{r,t,n} = z_{r,t} \quad \forall\, r \in R,\ \forall\, t \in D.$$

Interpretation:

- $z_{r,t} = 1$ se e solo se almeno un paziente occupa la stanza r il giorno t. I due vincoli a disuguaglianza ancorano $z_{r,t}$ al numero di pazienti effettivamente presenti.
- Se $z_{r,t} = 1$, la stanza r deve avere esattamente un infermiere assegnato il giorno t, vale a dire $\sum_n y_{r,t,n} = 1$. Se $z_{r,t} = 0$, nessun infermiere deve essere assegnato ($\sum_n y_{r,t,n} = 0$).

(6) Nurse Availability

$$y_{r,d,n} = 0 \quad \text{se } d \notin Shifts_n.$$

Interpretation: Un infermiere n può prestare servizio in una stanza r il giorno d solo se d rientra tra i giorni in cui l'infermiere è disponibile ($Shifts_n$).

(7) Maximum Number of Rooms per Nurse (Daily)

$$\sum_{r\in R} y_{r,d,n} \;\le\; 3 \quad \forall n \in N,\ \forall d \in D.$$

Interpretation: Ogni infermiere n non può essere assegnato a più di 3 stanze nello stesso giorno d, al fine di evitare un sovraccarico di lavoro.
Nel complesso, questi vincoli garantiscono che ciascun paziente venga ammesso una volta sola, che la capacità e le incompatibilità delle stanze siano rispettate, che ogni stanza occupata abbia un unico infermiere a disposizione, compatibilmente con i turni di lavoro e i limiti di assegnazione giornaliera del personale.

3 Q2: Pseudocode of the Algorithm

Provide a short pseudocode describing your algorithm.

3.1 Pseudocode of the Branch-and-Bound Algorithm

Below is a concise Branch-and-Bound procedure to solve the Mixed-Integer Linear Program

$$\begin{aligned} \min \quad & \sum_{i,r,d} \Big(d - \mathrm{Rel}[i]\Big)\, x_{i,r,d} \\ \text{subject to} \quad & \text{(capacity constraints, incompatibility, nurse availability, etc.)} \\ & x_{i,r,d},\ y_{r,d,n},\ z_{r,t} \ \in\ \{0,1\}. \end{aligned}$$

We adopt the style of *implicit enumeration* methods described in the course notes, of which Branch-and-Bound is a prime example.

3.2 Implicit Enumeration Methods (as in the Course Notes)

In the context of discrete optimization problems, such as Integer or Mixed-Integer Linear Programming, the term *implicit enumeration* refers to a class of techniques that systematically explore the entire space of solutions without explicitly listing all possible combinations. A pure *explicit enumeration* of every 0–1 assignment would be infeasible for large instances because the number of combinations grows exponentially.

By contrast, *Branch-and-Bound* and related algorithms (e.g. Branch-and-Cut) implicitly partition the search space:

- **Branch:** The feasible set is subdivided into smaller subproblems (or "nodes") by introducing additional constraints on a fractional variable found in the LP relaxation (e.g. forcing it to 0 or 1).
- **Bound:** Each subproblem inherits a bound from solving its *relaxation* (commonly a linear program). If the bound is already worse than the best known integer solution, we can *prune* that subproblem entirely, discarding all its (potentially huge) set of solutions.

This approach is called *implicit* because entire swaths of the solution space are eliminated via bounding without having to enumerate each combination. Despite this being a systematic search, it is still a form of partial enumeration that can reach the global optimum for the discrete problem.

3.3 Algorithmic Steps

Algorithm 1 Branch&Bound(P, *best_value*, *best_solution*)

Input: A MIP problem P (here, the patient–nurse–room assignment).
Initialize:
    best_value $\leftarrow +\infty$ // we want to minimize
    best_solution $\leftarrow$ `None`
    OpenNodes $\leftarrow \{P\}$
while *OpenNodes* is not empty **do**
  Take a subproblem P_k from *OpenNodes*.
  Solve the LP relaxation of P_k, ignoring the integrality constraints on (x, y, z).
  if LP relaxation is infeasible **or** the LP optimum $\geq$ best_value **then**
    Prune node P_k // no improvement possible
  else
    Let (x^*, y^*, z^*) be the LP solution for P_k; let obj = cost(x^*, y^*, z^*).
    if (x^*, y^*, z^*) is integral **then**
      if obj $<$ best_value **then**
        best_value $\leftarrow$ obj
        best_solution $\leftarrow (x^*, y^*, z^*)$
      end if
      Prune P_k // no need to branch further
    else
      // **Branch on a fractional variable** (e.g. some $x^*_{i,r,d}$ with $0 < x^*_{i,r,d} < 1$)
      Create two children:
        (a) $P_k^{(0)}$: add constraint $x_{i,r,d} = 0$
        (b) $P_k^{(1)}$: add constraint $x_{i,r,d} = 1$
      Insert $P_k^{(0)}$ and $P_k^{(1)}$ into *OpenNodes*.
    end if
  end if
end while
Output: *best_solution* and *best_value* (the final optimal MIP solution).

3.4 Criterion for Determining the Final Solution

- **Objective Function:** Minimize total admission delay, i.e.

$$\sum_{i,r,d} \Big(d - \text{Rel}[i]\Big)\, x_{i,r,d}.$$

- **Bounding:** Whenever the LP relaxation (with integrality relaxed) returns a cost $\geq$ *best_value*, we prune the node.

- **Integer Solution Check:** If the LP solution is *integral* and its cost improves upon *best_value*, then we update *best_value* and record this solution. We then prune the node, since no further branching on that node can yield a better solution.

- **Branching:** If the LP solution is fractional, choose a fractional variable (e.g. $x_{i,r,d}$) to branch on. One child sets $x_{i,r,d} = 0$, the other sets $x_{i,r,d} = 1$. Each child becomes a new subproblem.

At the end of this process, *best_solution* will be the optimal integer-feasible solution that achieves the minimum total delay. The bounding and branching steps ensure that we implicitly explore the search space in a systematic, yet efficient way.

3.4.1 VERSIONE SIMILE LIBRO

Below is a pseudocode that follows the same structure as in the provided snippet, using the same notation style. We assume a maximization problem for simplicity; in a minimization problem, signs and bound comparisons would be inverted accordingly.

Algorithm 2 BRANCH_AND_BOUND(P, v)

```
b = Bound( P, x, unfeasible )
if unfeasible = true then
    return −∞    // This subproblem has no feasible solutions
end if
if x is integer then
    return b    // The current solution is integral, return its value
else if b ≤ v then
    return −∞    // Cannot improve best known value
else
    // Branching step
    Branch(P, x, P_1, P_2, ..., P_k)
    for i = 1 to k do
        t = BRANCH_AND_BOUND(P_i, v)
        if t > v then
            v ← t
        end if
    end for
    return v
end if
```

Explanation of Key Steps:

- *Bound*$(P, x, \text{unfeasible})$ computes an upper bound b for subproblem P, and also returns a candidate solution x (possibly fractional) and a flag `unfeasible` that is `true` if the subproblem has no feasible solutions.
- If x is already an *integer* solution, the procedure returns the bound b (which in this branch-and-bound framework is simply the objective of the found integer solution).
- If $b \leq v$, no improvement is possible on the current best known value v, so the procedure returns $-\infty$ and effectively prunes this branch.

- Otherwise, the procedure *branches*, generating k new subproblems $P_1, \ldots, P_k$. Each of these is solved recursively, and any improved solution value t is used to update v.

Branch-and-Bound for Patient–Nurse–Room Assignment

Below is a high-level *Branch-and-Bound* pseudocode specialized to your healthcare scheduling project. It performs an explicit branch-and-bound search over the integrality constraints of the binary variables:

$$x_{i,r,d}, \quad y_{r,d,n}, \quad z_{r,t}.$$

At each node, the LP relaxation is solved; if the solution is integral, we compare it against the best known solution. Otherwise, we pick a fractional variable and create two child nodes.

4 Q3: Overview of the solution and computation

Report in a table the solutions you obtained (charger locations, value of the objective function, and computational time) for the five budget values you selected:

4.1 Solution Tables

Patient Assignments

Patient	Room	Admission Day	Length of Stay
0	3	0	3
1	2	0	2
2	4	0	4
3	2	0	3
4	1	1	2
5	4	1	3
6	1	0	1
7	0	2	2
8	3	0	3
9	2	2	3
10	3	3	2
11	0	4	2
12	0	4	1
13	1	1	2
14	0	2	3
15	1	3	1
16	0	0	2
17	1	4	3
18	0	5	2
19	0	0	3

Table 4: Patient-to-Room Assignments and Admission Days.

Nurse Assignments

Day	Room	Nurse
0	0	2
1	0	0
2	0	1
3	0	2
4	0	1
5	0	3
6	0	3
0	1	0
1	1	1
2	1	0
3	1	0
4	1	1
5	1	3
6	1	4
0	2	2
1	2	0
2	2	0
3	2	0
4	2	2
0	3	0
1	3	1
2	3	1
3	3	0
4	3	4
0	4	0
1	4	0
2	4	0
3	4	1

Table 5: Nurse-to-Room Assignments by Day.

5 Q4: Computational Complexity

Discuss the theoretical computational complexity of your algorithm:

- The solver usually use a branch-and bound strategy.
- The total number of $x_{i,r,d}$ variables (that defines if patient i starts his/her stay in room r on day d) is 700 since there are 20 patients, 5 rooms and 7 days => 20*5*7 = 700.
- The total number of $y_{i,r,d}$ variables (that defines if nurse n is assigned to room r on day d) is 175 since there are 5 rooms, 7 days and 5 nurses => 5*7*5 = 175.

- The size of the model (in terms of variables and constraints) grows polynomially with respect to the input parameters. But since the solver use branch-and-bound algorithm we split the problem in subproblems by assigning one variable to either 0 or 1. This technique reduce the number of nodes we explore but is not guaranteed that the execution time is polynomial. In the worst case scenario the time is exponential.

- The solution of an MILP requires, in the worst case, the exploration of a solution space that is exponential in the number of binary variables. In this case, the space is $2^{(|P|\times|R|\times|D|+|R|\times|D|\times|N|)}$. P patients, R rooms, D days, N nurses.

- **Polynomial complexity**. If B is the total number of binary variables (aka $|P| \times |R|\times|D|+|R|\times|D|\times|N|$ then the worst possible complexity is $O(2^B)$ this is because we might need to explore an exponential number of nodes.

1. General MIP is NP-Hard.
Integer and Mixed-Integer Linear Programming are well-known to be NP-hard. As such, there is no known polynomial-time algorithm that can solve any generic MIP problem to optimality in the worst case (unless P=NP).
2. Branch-and-Bound Worst-Case is Exponential.
Although Branch-and-Bound (B&B) can prune large portions of the search space, its worst-case performance remains exponential in the problem size. If pruning is ineffective for a particular instance, the algorithm might end up exploring a search tree whose size grows exponentially with respect to the input dimension.
3. Practical Performance.
Despite exponential worst-case behavior, modern MIP solvers often perform much better in practice. Powerful bounding strategies, cutting-plane methods, and various heuristics can drastically reduce the search tree, enabling the solver to handle many real-life problem instances within reasonable computing times.
Conclusion: From a purely theoretical standpoint, the Branch-and-Bound algorithm for our Mixed-Integer model has *exponential* complexity in the input size. This is fully consistent with the NP-hard nature of the MIP framework. However, due to pruning and other sophisticated techniques, the actual performance on many instances can be significantly more efficient than the worst-case exponential bound might suggest.

6 Q5: Exact or Heuristic?

Short Answer: Our Branch-and-Bound algorithm is an *exact* method, provided it completes the entire tree exploration (or proves optimality before exhaustively exploring every branch).

6.1 Why Is It Exact?

- **MILP modelling**: The problem is formulated as a mixed integer linear programming (MILP) model, where all decisions (e.g. allocation of patients to rooms and nurses to rooms on different days) are expressed via binary variables and constraints are formulated linearly.
- **Use of an Exact Solver**: The code uses the CBC solver via the mip library. MILP solvers such as CBC implement exact methods (typically branch-and-bound and branch-and-cut) that theoretically explore the solution space until optimality is guaranteed. If the solver is executed to completion, it provides an optimal solution or proves that there is no better solution.
- **Guarantee of Optimality:** Unlike heuristic approaches, which aim to find approximate solutions in reduced time without guaranteeing optimality, the method adopted here seeks exactly the best possible solution (minimising the admission delay) while respecting all constraints.
- **Complete Search (Implicit Enumeration).** In principle, the procedure systematically considers *every* feasible combination of integer variables. Pruning occurs only if a subproblem's lower bound (in a minimization context) is already worse than the best known solution, thus it cannot contain a better solution. This ensures no valid branch of the search is incorrectly discarded.
- **Finite Termination.** Once all relevant subproblems are either solved or pruned, the best integer-feasible solution found is globally optimal. By construction, the algorithm *implicitly enumerates* the solution space without actually listing every possible solution.
- **Contrast with Heuristics.** A *heuristic* might skip certain parts of the search space or make locally optimal "greedy" moves that cannot guarantee the global optimum. By contrast, a fully executed Branch-and-Bound algorithm *proves* optimality once no other subproblem can yield a better solution.
- **Conclusion:** The algorithm does not rely on approximations or techniques that sacrifice the quality of the solution for reduced execution time; on the contrary, it exactly defines and solves the optimisation problem through a MILP formulation and the use of an exact solver. Therefore, the algorithm is exact and not heuristic.

Hence, if the algorithm is not prematurely stopped and explores all necessary nodes, the solution it provides is guaranteed to be optimal.

Algorithm 3 BnB_HealthcareScheduling(*data_url*)

Input: A JSON file *data_url* with days, rooms, patients, nurses, etc.
Step 1: Load and Preprocess Data
1.1. Read JSON from *data_url*, parse into:
days, rooms, patients, nurses, ...
1.2. Extract parameters: capacities, release/due dates, incompatibilities, nurse-shift info.
Step 2: Initialize Branch&Bound
best_value $\leftarrow +\infty$ // Minimizing total admission delay
best_solution $\leftarrow$ None
OpenNodes $\leftarrow$ {root_problem(P)} // The original MIP with no extra branching constraints
Step 3: Main B&B Loop
while *OpenNodes* is not empty **do**
3.1. Pick a node P_k from *OpenNodes*.
3.2. Build the `Model` for P_k:
- Add constraints:
(a) Unique Admission: $\sum_{r,d} x_{i,r,d} = 1,\ \forall i$.
(b) Capacity: $\sum_{i,d\,:\,d \le t < d+L_i} x_{i,r,d} \le \text{Cap}[r]$.
(c) Incompatibility: $x_{i,r,d} = 0$ if $r \in \text{Incomp}[i]$.
(d) Admission Window: $x_{i,r,d} = 0$ if $d < \text{Rel}[i]$ or $d > \text{Due}[i]$.
(e) Auxiliary $z_{r,t}$ to indicate if room r is occupied on day t.
(f) Exactly 1 Nurse per Occupied Room: $\sum_n y_{r,t,n} = z_{r,t}$.
(g) Nurse availability: $y_{r,d,n} = 0$ if $d \notin \text{Shifts}[n]$.
(h) Each nurse ≤ 3 rooms/day: $\sum_r y_{r,d,n} \le 3$.
- Add *branching constraints* (e.g. $x_{i^*,r^*,d^*} = 0/1$) from parent nodes.
- **Objective:** min $\sum_{i,r,d}(d - \text{Rel}[i])x_{i,r,d}$.
- Solve the LP relaxation (ignoring integrality).
3.3. **Bounding & Pruning**:
if LP is infeasible **or** LP optimum $\ge$ best_value **then**
Prune P_k // No improvement possible
else
Let (x^*, y^*, z^*) be LP solution; obj = cost(x^*, y^*, z^*)
if (x^*, y^*, z^*) is integral **then**
if obj $<$ best_value **then**
best_value $\leftarrow$ obj
best_solution $\leftarrow (x^*, y^*, z^*)$
end if
Prune P_k // integral solution found
else
// **Branch** on a fractional variable (e.g. x_{i^*,r^*,d^*})
Create children:
$P_k^{(0)}$: $x_{i^*,r^*,d^*} = 0$
$P_k^{(1)}$: $x_{i^*,r^*,d^*} = 1$
Insert both children into *OpenNodes*.
end if
end if
end while
Output: *best_solution* (with minimal admission delay *best_value*).