BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC PHENIKAA**

---


**CƠ SỞ ĐỒ ÁN**

# NGHIÊN CỨU VỀ CÔNG CỤ HỖ TRỢ OR-TOOLS GOOGLE VỀ TỐI UY HÓA

**Giảng viên hướng dẫn:** **TS. Nguyễn Thanh Hoàng**

**Hoàng Đức Cường:** **MSSV: 20010941**

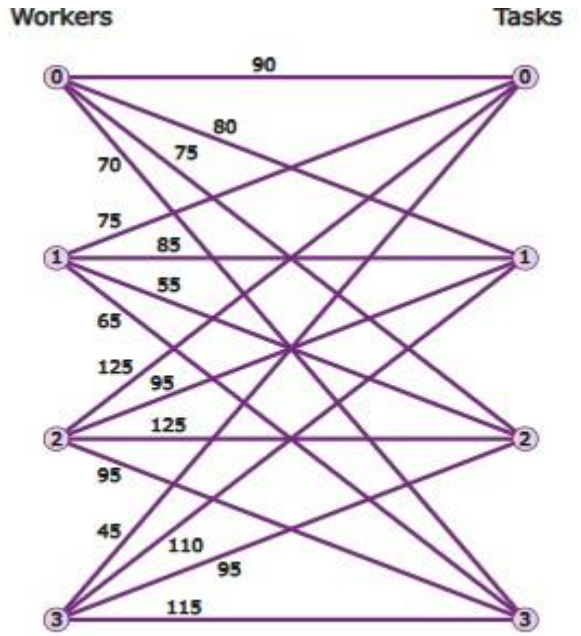
# BÀI LÀM

## I) Giới thiệu bài toán và công cụ

### 1) Tổng quan

-Ngày nay có rất nhiều bài toán tối uy được sử dụng như một cách thực hiện những công việc khó khăn ngoài cuộc sống, đặc biệt nó giúp giải quyết những bài toán lớn trong thực tế cuộc sống hiện nay. Nổi tiếng trong số đó là bài toán Gán để giải quyết những bài toán tổ hợp cơ bản diễn ra trong thực tế cuộc sống mà công cụ hỗ trợ or-tools được phát triển bằng google sẽ giúp ta giải quyết một cách nhanh tróng và dễ dàng hơn, đồng thời giúp ta có thể nâng cấp bài toán để giải quyết những vấn đề khó khăn hơn ngoài cuộc sống.

-Ví dụ một công nhân cần thực hiện nhiệm vụ trong một khoảng thời gian nhất định ta cần đảm bảo người đó có thể hoàn thành công việc một cách nhanh nhất để đảm bảo tiến độ công việc ngoài thực tế. Bài toán đặt ra chỉ định mỗi công nhân làm nhiều nhất một nhiệm vụ đồng thời giảm thiểu tổng chi phí để có thể hoàn thành công việc




-Qua sơ đồ này ta thấy được có 4 công nhân và 4 nhiệm vụ, các cạnh đại diện cho những cách phân chia công việc trên các cạnh là thời gian để công nhân có thể hoàn thành công việc. Qua đó ta phải tính được thời gian ngắn nhất mỗi công nhân có thể làm nhiệm vụ của mình để hoàn thành công việc trong thời gian ngắn nhất.

-Công cụ OR-TOOLS cung cấp cho ta hai bộ giải vô cùng tối ưu đó là bộ giải MIP và bộ giải CP-SAT

2) Giới thiệu công cụ

-Phương pháp xây dựng và giải mô hình CP-SAT.Phương pháp xây dựng và giải mô hình CP-SAT.

-CpModel: Các phương thức tạo mô hình, bao gồm các biến và các ràng buộc.

-CPSolver: Phương pháp giải một mô hình và đánh giá các giải pháp.

-CpSolverSolutionCallback: Một phương pháp chung để thực hiện các cuộc gọi lại.

-ObjectiveSolutionPrinter: In các giá trị mục tiêu và thời gian đã trôi qua cho các giải pháp trung gian.

-VarArraySolutionPrinter: In nghiệm trung gian (giá trị biến, thời gian).

-Constraint: Một vài phương thức tiện ích để sửa đổi các giới hạn được tạo bởi CpModel.

-LinearExpr: Phương pháp tạo các ràng buộc và mục tiêu từ các mảng hệ số lớn.

-Class CpModel

+New: tạo các biến số nguyên, boolean hoặc khoảng.

+Add: tạo các ràng buộc mới và thêm chúng vào mô hình.

II) Giải bài tập

1) Bài 1: Phân chia công việc theo bảng sau(Yêu cầu mỗi công nhân chỉ thực hiện 1 nhiệm vụ sao cho thời gian là ngắn nhất để hoàn thành công việc)

| Người làm việc | nhiệm vụ 0 | Nhiệm vụ 1 | Nhiệm vụ 2 | nhiệm vụ 3 |
|---|---|---|---|---|
| 0 | 90 | 80 | 75 | 70 |
| 1 | 35 | 85 | 55 | 65 |
| 2 | 125 | 95 | 90 | 95 |
| 3 | 45 | 110 | 95 | 115 |
| 4 | 50 | 100 | 90 | 100 |

-Với bài toán này ta sẽ giải theo trình giải MIP

Bước 1 ta khai báo thư việc cho bài toán

from ortools.linear_solver import pywraplp

Bước 2 ta tạo cho bài toán dữ liệu(data cần thiết)

Bước 3: Khai báo trình giải MIP

Bước 4: Tạo các rằng buộc cho bài toán

Bước 5: Tạo các hàm mục tiêu cho bài toán

Bước 6 goi ra bộ giải
status = solver.Solve()

Bước 7: In ra kết quả hoàn thành bài toán

⇨ Sau đây là lời giải cho bài toán

```python
from ortools.linear_solver import pywraplp
def main():
    costs = [
        [90, 80, 75, 70],
        [35, 85, 55, 65],
        [125, 95, 90, 95],
        [45, 110, 95, 115],
        [50, 100, 90, 100],
    ]
    num_workers = len(costs)
    num_tasks = len(costs[0])
    solver = pywraplp.Solver.CreateSolver('SCIP')
    if not solver:
        return
    # Variables
    # x[i, j] is an array of 0-1 variables, which will be 1
    # if worker i is assigned to task j.
    x = {}
    for i in range(num_workers):
        for j in range(num_tasks):
            x[i, j] = solver.IntVar(0, 1, '')
    # Constraints
    # Each worker is assigned to at most 1 task.
    for i in range(num_workers):
        solver.Add(solver.Sum([x[i, j] for j in range(num_tasks)]) <= 1)
    # Each task is assigned to exactly one worker.
    for j in range(num_tasks):
        solver.Add(solver.Sum([x[i, j] for i in range(num_workers)]) == 1)
    # Objective
    objective_terms = []
    for i in range(num_workers):
        for j in range(num_tasks):
            objective_terms.append(costs[i][j] * x[i, j])
    solver.Minimize(solver.Sum(objective_terms))
    # Solve
    status = solver.Solve()
    # Print solution.
    if status == pywraplp.Solver.OPTIMAL or status == pywraplp.Solver.FEASIBLE:
        print(f'Total cost = {solver.Objective().Value()}\n')
        for i in range(num_workers):
            for j in range(num_tasks):
                # Test if x[i,j] is 1 (with tolerance for floating point arithmetic).
                if x[i, j].solution_value() > 0.5:
                    print(f'Worker {i} assigned to task {j}.' +
                          f' Cost: {costs[i][j]}')
    else:
        print('No solution found.')
```

```
Total cost = 265.0

Worker 0 assigned to task 3. Cost: 70
Worker 1 assigned to task 2. Cost: 55
Worker 2 assigned to task 1. Cost: 95
Worker 3 assigned to task 0. Cost: 45
PS D:\Download\SmartHome>
```

Hàm costs =[…]
->tạo data cho bài toán
num_workers = len(costs)
num_tasks = len(costs[0])
->tạo chuỗi cho số lượng nhân viên và nhiệm vụ
x = {}
    for i in range(num_workers):
        for j in range(num_tasks):
            x[i, j] = solver.IntVar(0, 1, '')
Tạo giá trị x duyệt số lượng công nhân và nhiệm vụ theo i và j
X[i, j]  số lương công nhân i thực hiện nhiệm vụ j

```
for i in range(num_workers):
        solver.Add(solver.Sum([x[i, j] for j in range(num_tasks)]) <= 1)
```
Mỗi công nhân i thực hiện nhiều nhất 1 nhiệm vụ j

```
for j in range(num_tasks):
        solver.Add(solver.Sum([x[i, j] for i in range(num_workers)]) == 1)
```
Mỗi nhiệm vụ được đảm bảo giao cho 1 công nhân

```
objective_terms = []
    for i in range(num_workers):
        for j in range(num_tasks):
            objective_terms.append(costs[i][j] * x[i, j])
    solver.Minimize(solver.Sum(objective_terms))
```
Duyệt và thêm vào hàm mục tiêu đảm bảo thời gian * x[i,j]
Hàm minimize(solver.Sum) đảm bảo hàm mục tiêu nhỏ nhất

```
for i in range(num_workers):
            for j in range(num_tasks):
                if x[i, j].solution_value() > 0.5:
                        print(f'Worker {i} assigned to task {j}.' +
                                f' Cost: {costs[i][j]}')
```

Nếu giá trị bài toán ra lẻ >0.5 thì sẽ làm tròn lên 1 đơn vị đồng thời in ra kết quả bài toán

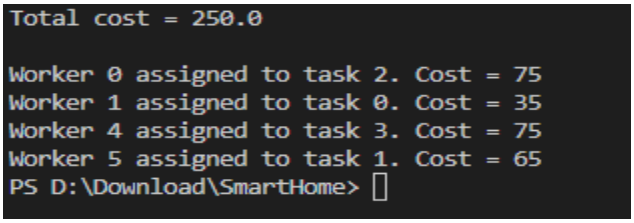
2) Bài toán 2(Nâng cấp lên với nhiệm vụ cho 1 nhóm công nhân)Ta có 4 nhiệm vụ nhưng có để tận 6 nhân viên chia làm 2 đội và mỗi đội chỉ có thể thực hiện 2 nhiệm vụ

➔ Lần này chúng ta sử dụng CP-SAT

Bước 1: Nhập thư viện cần thiết
model = cp_model.CpModel()
Bước 2: tạo data và tạo ra 2 team mỗi team thực hiện 2 nhiệm vụ
Bước 3: Tạo mô hình giải mã
Bước 4: Tạo các biến cho bài toán
Bước 5: Thêm các rằng buộc cho bài toán
Bước 6: Tạo hàm mục tiêu giải quyết bài toán
Bước 7: gọi bộ giải
solver = cp_model.CpSolver()
status = solver.Solve(model)
Bước 8: in ra bài toán

⇨ Sau đây là lời giải cho bài toán

```python
from ortools.sat.python import cp_model
def main():
    costs = [
        [90, 76, 75, 70],
        [35, 85, 55, 65],
        [125, 95, 90, 105],
        [45, 110, 95, 115],
        [60, 105, 80, 75],
        [45, 65, 110, 95],
    ]
    num_workers = len(costs)
    num_tasks = len(costs[0])
    team1 = [0, 2, 4]
    team2 = [1, 3, 5]
    team_max = 2
    model = cp_model.CpModel()
    x = {}
    for worker in range(num_workers):
        for task in range(num_tasks):
            x[worker, task] = model.NewBoolVar(f'x[{worker},{task}]')
    for worker in range(num_workers):
        model.AddAtMostOne(x[worker, task] for task in range(num_tasks))
    for task in range(num_tasks):
        model.AddExactlyOne(x[worker, task] for worker in range(num_workers))
    team1_tasks = []
    for worker in team1:
        for task in range(num_tasks):
            team1_tasks.append(x[worker, task])
    model.Add(sum(team1_tasks) <= team_max)

    team2_tasks = []
    for worker in team2:
        for task in range(num_tasks):
            team2_tasks.append(x[worker, task])
    model.Add(sum(team2_tasks) <= team_max)
    objective_terms = []
    for worker in range(num_workers):
        for task in range(num_tasks):
            objective_terms.append(costs[worker][task] * x[worker, task])
    model.Minimize(sum(objective_terms))
    solver = cp_model.CpSolver()
    status = solver.Solve(model)
    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        print(f'Total cost = {solver.ObjectiveValue()}\n')
        for worker in range(num_workers):
            for task in range(num_tasks):
                if solver.BooleanValue(x[worker, task]):
                    print(f'Worker {worker} assigned to task {task}.' +
                          f' Cost = {costs[worker][task]}')
```




-Giải thích code

team1 = [0, 2, 4]
team2 = [1, 3, 5]
team_max = 2
Chia nhóm thành 2: nhóm 1: Công nhân 0,2,4; Nhóm 2 công nhân 1,3,5
Tổng số nhiệm vụ tối đa cho bất kỳ nhóm nào
x = {}
for worker in range(num_workers):
    for task in range(num_tasks):
        x[worker, task] = model.NewBoolVar(f'x[{worker},{task}]')
Tạo giá trị x duyệt số lượng công nhân và nhiệm vụ theo worker và task
X[worker, task] số lượng công nhân thực hiện nhiệu vụ tuy nhiên không giống bài toán 1 nên các nút sẽ được đánh số khác nhau theo yêu cầu của bộ giải

for worker in range(num_workers):
    model.AddAtMostOne(x[worker, task] for task in range(num_tasks))
Mỗi công nhân được giao nhiều nhất một nhiệm vụ

for task in range(num_tasks):
    model.AddExactlyOne(x[worker, task] for worker in range(num_workers))
Mỗi nhiệm vụ được giao cho chính xác một công nhân.

for worker in team1:
    for task in range(num_tasks):
        team1_tasks.append(x[worker, task]) // thêm
model.Add(sum(team1_tasks) <= team_max)

team2_tasks = []
for worker in team2:
    for task in range(num_tasks):
        team2_tasks.append(x[worker, task]) //thêm
model.Add(sum(team2_tasks) <= team_max)
Mỗi đội đảm nhận nhiều nhất hai nhiệm vụ.
Tiếp theo là gọi hàm mục tiêu, gọi bộ giải và in ra kết quả bài toán

3) Bài toán 3 Bài tập với kích thước nhiệm vụ biểu thị lượng thời gian mà nhiệm vụ và Tổng kích thước của các nhiệm vụ được thực hiện bởi mỗi công nhân có giới hạn cố định

⇨ Sử dụng bộ giải MIP

Bước 1 Khai báo thư viện

```
from ortools.sat.python import cp_model
```

Bước 2 tạo data cho bài toán, tạo kích thước cho nhiệm vụ, tạo giới hạn trên của tổng kích thước các nhiệm vụ được thực hiện bởi bất kỳ công nhân đơn lẻ nào.

Bước 3: Khai báo bộ giải

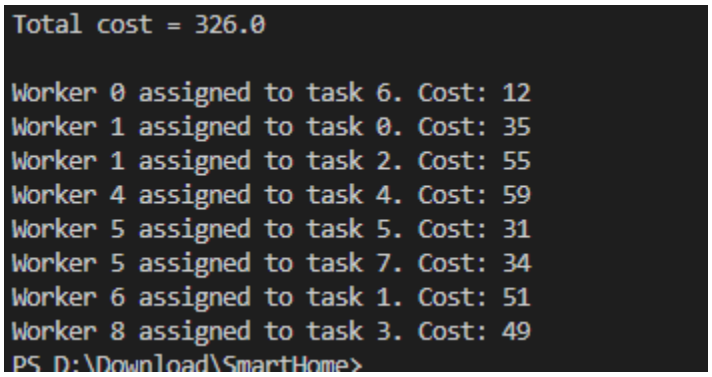
Bước 4: Tạo các biến

Bước 5: Thêm các rằng buộc cho bài toán

Bước 6: Tạo ra hàm mục tiêu

Bước 7 In ra kết quả bài toán và làm tròn số

```python
from ortools.linear_solver import pywraplp
def main():
    costs = [
        [90, 76, 75, 70, 50, 74, 12, 68],
        [35, 85, 55, 65, 48, 101, 70, 83],
        [125, 95, 90, 105, 59, 120, 36, 73],
        [45, 110, 95, 115, 104, 83, 37, 71],
        [60, 105, 80, 75, 59, 62, 93, 88],
        [45, 65, 110, 95, 47, 31, 81, 34],
        [38, 51, 107, 41, 69, 99, 115, 48],
        [47, 85, 57, 71, 92, 77, 109, 36],
        [39, 63, 97, 49, 118, 56, 92, 61],
        [47, 101, 71, 60, 88, 109, 52, 90],
    ]
    num_workers = len(costs)
    num_tasks = len(costs[0])
    task_sizes = [10, 7, 3, 12, 15, 4, 11, 5]
    total_size_max = 15
    solver = pywraplp.Solver.CreateSolver('SCIP')
    if not solver:
        return
    x = {}
    for worker in range(num_workers):
        for task in range(num_tasks):
            x[worker, task] = solver.BoolVar(f'x[{worker},{task}]')
    for worker in range(num_workers):
        solver.Add(
            solver.Sum([
                task_sizes[task] * x[worker, task] for task in range(num_tasks)
            ]) <= total_size_max)
    for task in range(num_tasks):
        solver.Add(
            solver.Sum([x[worker, task] for worker in range(num_workers)]) == 1)
    objective_terms = []
    for worker in range(num_workers):
        for task in range(num_tasks):
            objective_terms.append(costs[worker][task] * x[worker, task])
    solver.Minimize(solver.Sum(objective_terms))
    status = solver.Solve()
    if status == pywraplp.Solver.OPTIMAL or status == pywraplp.Solver.FEASIBLE:
        print(f'Total cost = {solver.Objective().Value()}\n')
        for worker in range(num_workers):
            for task in range(num_tasks):
                if x[worker, task].solution_value() > 0.5:
                    print(f'Worker {worker} assigned to task {task}.' +
                          f' Cost: {costs[worker][task]}')
    else:
        print('No solution found.')
```




task_sizes = [10, 7, 3, 12, 15, 4, 11, 5]
total_size_max = 15
kích cỡ của mỗi nhiệm vụ: lần lượt 10, 7, 3, 12, 15, 4, 11,5
vì có 8 nhiệm vụ mà có 10 công nhân nên có số lượng công nhân thừa
Tổng kích thước nhiệm vụ tối đa cho bất kỳ công nhân nào

```
for worker in range(num_workers):
    solver.Add(
        solver.Sum([
            task_sizes[task] * x[worker, task] for task in range(num_tasks)
]) <= total_size_max)
```

Tổng kích thước nhiệm vụ mà mỗi nhân viên đảm nhận tối đa là total_size_max.

```
for task in range(num_tasks):
    solver.Add(
        solver.Sum([x[worker, task] for worker in range(num_workers)]) == 1)
```

Mỗi nhiệm vụ được giao cho chính xác một công nhân.

Tiếp theo là tạo hàm mục tiêu và gọi bộ giải và in ra kết quả bài toán

4) Bài toán 4: Bài tập các nhóm được phép ( một vấn đề về phân công công việc trong đó có 12 công nhân hoàn thành các công việc được phân chia thành 3 nhóm được đánh số từ 0-11 và các nhóm đuược phép là sự kết hợp của các cặp công nhân
-> giải pháp
-để xác định nhóm công nhân được phép thì sử dụng bộ giải CP-SAT tạo các mảng nhị phân cho biết công nhận nào thuộc các nhóm tạo thành các vector như hình

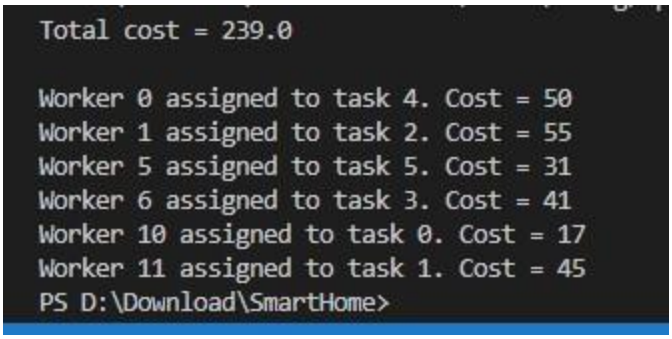
```
group1 = [
    [0, 0, 1, 1],  # Workers 2, 3
    [0, 1, 0, 1],  # Workers 1, 3
    [0, 1, 1, 0],  # Workers 1, 2
    [1, 1, 0, 0],  # Workers 0, 1
    [1, 0, 1, 0],  # Workers 0, 2
]

group2 = [
    [0, 0, 1, 1],  # Workers 6, 7
    [0, 1, 0, 1],  # Workers 5, 7
    [0, 1, 1, 0],  # Workers 5, 6
    [1, 1, 0, 0],  # Workers 4, 5
    [1, 0, 0, 1],  # Workers 4, 7
]

group3 = [
    [0, 0, 1, 1],  # Workers 10, 11
    [0, 1, 0, 1],  # Workers 9, 11
    [0, 1, 1, 0],  # Workers 9, 10
    [1, 0, 1, 0],  # Workers 8, 10
    [1, 0, 0, 1],  # Workers 8, 11
]
```

Nhóm công nhân kết hợp  bằng 1 và không được chọn sẽ nhận giá trị bằng 0
-Các bước để giải bài toán
Bước 1: Khai báo thư viện
Bước 2: Xác định dữ liệu
Bước 3: Tạo các nhóm được phép
Bước 4: Tạo mô hình
Bước 5: Tạo các biến
Bước 6 thêm các ràng buộc
Bước 7: Tạo hàm mục tiêu
Bước 8: Gọi bộ giải
Bước 9: In và giải ra kết quả

->Sau đây là cách giải

```python
from ortools.sat.python import cp_model


def main():
    costs = [
        [90, 76, 75, 70, 50, 74],
        [35, 85, 55, 65, 48, 101],
        [125, 95, 90, 105, 59, 120],
        [45, 110, 95, 115, 104, 83],
        [60, 105, 80, 75, 59, 62],
        [45, 65, 110, 95, 47, 31],
        [38, 51, 107, 41, 69, 99],
        [47, 85, 57, 71, 92, 77],
        [39, 63, 97, 49, 118, 56],
        [47, 101, 71, 60, 88, 109],
        [17, 39, 103, 64, 61, 92],
        [101, 45, 83, 59, 92, 27],
    ]
    num_workers = len(costs)
    num_tasks = len(costs[0])
    group1 = [
        [0, 0, 1, 1],  # Workers 2, 3
        [0, 1, 0, 1],  # Workers 1, 3
        [0, 1, 1, 0],  # Workers 1, 2
        [1, 1, 0, 0],  # Workers 0, 1
        [1, 0, 1, 0],  # Workers 0, 2
    ]

    group2 = [
        [0, 0, 1, 1],  # Workers 6, 7
        [0, 1, 0, 1],  # Workers 5, 7
        [0, 1, 1, 0],  # Workers 5, 6
        [1, 1, 0, 0],  # Workers 4, 5
        [1, 0, 0, 1],  # Workers 4, 7
    ]

    group3 = [
        [0, 0, 1, 1],  # Workers 10, 11
        [0, 1, 0, 1],  # Workers 9, 11
        [0, 1, 1, 0],  # Workers 9, 10
        [1, 0, 1, 0],  # Workers 8, 10
        [1, 0, 0, 1],  # Workers 8, 11
    ]
```

```
    model = cp_model.CpModel()
    x = {}
    for worker in range(num_workers):
        for task in range(num_tasks):
            x[worker, task] = model.NewBoolVar(f'x[{worker},{task}]')
    for worker in range(num_workers):
        model.AddAtMostOne(x[worker, task] for task in range(num_tasks))
    for task in range(num_tasks):
        model.AddExactlyOne(x[worker, task] for worker in range(num_workers))
    work = {}
    for worker in range(num_workers):
        work[worker] = model.NewBoolVar(f'work[{worker}]')
    for worker in range(num_workers):
        for task in range(num_tasks):
            model.Add(work[worker] == sum(
                x[worker, task] for task in range(num_tasks)))
    model.AddAllowedAssignments([work[0], work[1], work[2], work[3]], group1)
    model.AddAllowedAssignments([work[4], work[5], work[6], work[7]], group2)
    model.AddAllowedAssignments([work[8], work[9], work[10], work[11]], group3)
    objective_terms = []
    for worker in range(num_workers):
        for task in range(num_tasks):
            objective_terms.append(costs[worker][task] * x[worker, task])
    model.Minimize(sum(objective_terms))
    solver = cp_model.CpSolver()
    status = solver.Solve(model)
    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        print(f'Total cost = {solver.ObjectiveValue()}\n')
        for worker in range(num_workers):
            for task in range(num_tasks):
                if solver.BooleanValue(x[worker, task]):
                    print(f'Worker {worker} assigned to task {task}.' +
                          f' Cost = {costs[worker][task]}')
    else:
        print('No solution found.')


if __name__ == '__main__':
    main()
```




Giải thích code

```
model.AddAllowedAssignments([work[0], work[1], work[2], work[3]], group1)
model.AddAllowedAssignments([work[4], work[5], work[6], work[7]], group2)
model.AddAllowedAssignments([work[8], work[9], work[10], work[11]], group3)
```

Các biến work[i]là các biến 0-1 cho biết trạng thái công việc hoặc từng công nhân. Work[i] bằng 1 nếu nhân viên i được giao cho một nhiệm vụ và bằng 0 nếu ngược lại. Solver.Add(solver.AllowedAssignments([work[0], work[1], work[2], work[3]], group1)) xác định ràng buộc rằng trạng thái công việc của công nhân 0 - 3 phải khớp với một trong các mẫu trong nhóm 1, tương tự như thế với các nhóm 2, 3

```
for worker in range(num_workers):
    model.AddAtMostOne(x[worker, task] for task in range(num_tasks))
```

Mỗi công nhân được chỉ đỉnh nhiều nhất 1 nhiệm vụ

```
for task in range(num_tasks):
    model.AddExactlyOne(x[worker, task] for worker in range(num_workers))
```

Mỗi nhiệm vụ được gán cho chính xác một công nhân

```
for worker in range(num_workers):
        work[worker] = model.NewBoolVar(f'work[{worker}]')

    for worker in range(num_workers):
        for task in range(num_tasks):
            model.Add(work[worker] == sum(
                x[worker, task] for task in range(num_tasks)))
```

Tạo các biến cho mỗi nhân viên, cho biết họ có làm việc trên một số tác vụ hay không.

5) Bài toán 5: Giải bài toán chi phí và nhiệm vụ của bài toán 1 bằng phương pháp tổ hợp tổng tuyến tính

```python
import numpy as np
from ortools.graph.python import linear_sum_assignment
def main():
    """Linear Sum Assignment example."""
    assignment = linear_sum_assignment.SimpleLinearSumAssignment()

    costs = np.array([
        [90, 76, 75, 70],
        [35, 85, 55, 65],
        [125, 95, 90, 105],
        [45, 110, 95, 115],
    ])
    end_nodes_unraveled, start_nodes_unraveled = np.meshgrid(
        np.arange(costs.shape[1]), np.arange(costs.shape[0]))
    start_nodes = start_nodes_unraveled.ravel()
    end_nodes = end_nodes_unraveled.ravel()
    arc_costs = costs.ravel()

    assignment.add_arcs_with_cost(start_nodes, end_nodes, arc_costs)

    status = assignment.solve()

    if status == assignment.OPTIMAL:
        print(f'Total cost = {assignment.optimal_cost()}\n')
        for i in range(0, assignment.num_nodes()):
            print(f'Worker {i} assigned to task {assignment.right_mate(i)}.' +
                  f'  Cost = {assignment.assignment_cost(i)}')
    elif status == assignment.INFEASIBLE:
        print('No assignment is possible.')
    elif status == assignment.POSSIBLE_OVERFLOW:
        print(
            'Some input costs are too large and may cause an integer overflow.')

if __name__ == '__main__':
    main()
```

Các bước giải
Bước 1 nhập thư viện:  dòng 1 đến 2
Bước 2 Xác định dữ liệu dòng 7 đến 17
Tạo ra 1 mảng ma trận chi phí có i,j là chi phí thực hiện nhiệm vụ j của công nhân i
Bước 3: Tạo bộ giải dòng 5
Bước 4: Thêm rang buộc dòng 19( thêm chi phí vào bộ giải bằng cách lặp qua các công nhân và tác vụ
Bước 5: Gọi bộ giả: dòng 21

Bước 6: In ra kết quả bài toán bằng cách kiểm tra tối ưu vì thuật toán này chỉ giải quyết được dữ liệu ko quá lớn nên kiểm xem có nhiệm vụ không với nhiệm vụ và data có quá lớn hay không dòng 23 đến 32

➔ Nhận thấy kết quả giống bài 1 nên khi có bài toán cơ bản nên dùng Linear Sum Assignment Solver

Bài toán 7 bổ xung bài toán 1 nếu muốn đặt nhiều số công nhân hoàn thành nhiệm vụ thì ta có thể dùng hàm limit

```
def main():
    # Data
    costs = [
        [90, 80, 75, 70],
        [35, 85, 55, 65],
        [125, 95, 90, 95],
        [45, 110, 95, 115],
        [50, 100, 90, 100],
        [47, 115, 25, 48],
    ]
    limit = [2, 1, 2, 1]
    num_workers = len(costs)
    num_tasks = len(costs[0])

    # Solver
    # Create the mip solver with the SCIP backend.
    solver = pywraplp.Solver.CreateSolver('SCIP')

    if not solver:
        return

    # Variables
    # x[i, j] is an array of 0-1 variables, which will be 1
    # if worker i is assigned to task j.
    x = {}
    for i in range(num_workers):
        for j in range(num_tasks):
            x[i, j] = solver.IntVar(0, 1, '')

    # Constraints
    # Each worker is assigned to at most 1 task.
    for i in range(num_workers):
        solver.Add(solver.Sum([x[i, j] for j in range(num_tasks)]) <= 1)

    # Each task is assigned to exactly one worker.
    for j in range(num_tasks):
        solver.Add(solver.Sum([x[i, j] for i in range(num_workers)]) == limit[j])
```

Nếu ta xét như trên sẽ có 2 người làm nhiệm vụ 0 và 2 người làm nhiệm vụ 2

```
Total cost = 340.0

Worker 0 assigned to task 3. Cost: 70
Worker 1 assigned to task 2. Cost: 55
Worker 2 assigned to task 1. Cost: 95
Worker 3 assigned to task 0. Cost: 45
Worker 4 assigned to task 0. Cost: 50
Worker 5 assigned to task 2. Cost: 25
```

III) Nghiên cứu thêm

1) Thuật toán quay lui (Backtracking)

-Quay lui là một kĩ thuật thiết kế giải thuật dựa trên đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn khả dĩ và đệ quy. Người đầu tiên đề ra thuật ngữ này (backtrack) là nhà toán học người Mỹ D. H. Lehmer vào những năm 1950.

-Tư tưởng: Dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng.

Mô hình thuật toán

```
Backtracking(k) {
        for([Mỗi phương án chọn i(thuộc tập D)]) {
                if ([Chấp nhận i]) {
                        [Chọn i cho X[k]];
                        if ([Thành công]) {
                                [Đưa ra kết quả];
                        } else {
                                Backtracking(k+1);
                                [Bỏ chọn i cho X[k]];
                        }
                }
        }
}
```

Nhận xét

-Ưu điểm: Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy.

Nhược điểm:

+Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ. Vì nó mắc phải các nhược điểm sau:

+Rơi vào tình trạng "thrashing": qúa trình tìm kiếm cứ gặp phải bế tắc với cùng một nguyên nhân.

+Thực hiện các công việc dư thừa: Mỗi lần chúng ta quay lui, chúng ta cần phải đánh giá lại lời giải trong khi đôi lúc điều đó không cần thiết.

+Không sớm phát hiện được các khả năng bị bế tắc trong tương lai. Quay lui chuẩn, không có cơ chế nhìn về tương lai để nhận biết đc nhánh tìm kiếm sẽ đi vào bế tắc.

2) Constraint programming(lập trình ràng buộc) CP là tên được đặt để xác định các giải pháp khả thi trong số rất nhiều ứng cử viên, trong đó vấn đề có thể được mô hình hóa theo các ràng buộc tùy ý. Các vấn đề về CP nảy sinh trong nhiều ngành khoa học và kỹ thuật.
-CP dựa trên tính khả thi (tìm giải pháp khả thi) hơn là tối ưu hóa (tìm giải pháp tối ưu) và tập trung vào các ràng buộc và biến hơn là hàm mục tiêu. Trên thực tế, một vấn đề CP thậm chí có thể không có hàm mục tiêu — mục tiêu có thể chỉ đơn giản là thu hẹp một tập hợp rất lớn các giải pháp khả thi thành một tập hợp con dễ quản lý hơn bằng cách thêm các ràng buộc vào vấn đề.
-Một ví dụ về vấn đề rất phù hợp với CP là lập kế hoạch cho nhân viên . Vấn đề phát sinh khi các công ty hoạt động liên tục — chẳng hạn như các nhà máy — cần tạo lịch trình hàng tuần cho nhân viên của họ. Đây là một ví dụ rất đơn giản: một công ty chạy ba ca 8 giờ mỗi ngày và phân công ba trong số bốn nhân viên của mình vào các ca khác nhau mỗi ngày, đồng thời cho người thứ tư nghỉ vào ngày đó. Ngay cả trong một trường hợp nhỏ như vậy, số lượng lịch trình có thể có là rất lớn: mỗi ngày, có $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$ nhiệm vụ nhân viên có thể, vì vậy số lịch tuần có thể có là 24 7, tức là hơn 4,5 tỷ. Thông thường sẽ có những ràng buộc khác làm giảm số lượng giải pháp khả thi — ví dụ: mỗi nhân viên phải làm việc ít nhất một số ngày tối thiểu mỗi tuần. Phương pháp CP theo dõi những giải pháp nào vẫn khả thi khi bạn thêm các ràng buộc mới, điều này làm cho nó trở thành một công cụ mạnh mẽ để giải quyết các vấn đề lập lịch trình lớn trong thế giới thực.

III) Kết quả
- Đa phần đã nắm được chức năng của công cụ or tools tiêu biểu là hai công cụ hỗ trợ MIPS và CP SAT
- Hiểu được sự cần thiết của công cụ đối với các bài toán về tối ưu để giải quyết những vấn đề của thực tiễn đời sống
- Tìm hiểu các bài toán có sẵn để nghiên cứu về những bài toán khó khăn thực tiễn hơn trong tương lai
- Còn chưa hiểu được hết các hàm trong thuật toán tối uy (sẽ khắc phục hơn trong tương lai)
-chưa hiểu rõ MIPS và CP SAT thuật toán nào tối ưu trong từng trường hợp
- link github: https://github.com/HoangDucCuong2002/DOAN_ORTOOLS