VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

CS106 KHTN

Báo cáo bài tập Sokoban_v2

Sinh Viên :
Đồng Quốc Thắng

Giảng viên :
Lương Ngọc Hoàng

Ngày 16 tháng 4 năm 2025

Mục lục

Chương 1

Hướng giải quyết và code của bài tập

Code của em cho bài tập này và file kết quả .csv có ở:
`https://github.com/LowTechTurtle/CS106-AI/tree/main/knapsack`

1.1 Hướng giải quyết

Trong bài tập phần này em đã chọn: 5 test trong mỗi group(có 13 group được đánh dấu từ 00 tới 12). Các test có size từ 100, 200 và 500.

Mỗi test sẽ chạy nhiều nhất là 60 giây, nếu sau 60 giây mà vẫn chưa tìm ra được kết quả tối ưu thì sẽ timeout và đưa ra kết quả đã có hiện tại

Trong file kết quả sẽ có 8 cột, ý nghĩa của từng cột như sau:

- group: 1 trong 13 group test
- file: file nào ở trong group đó
- value: điểm lợi ích tìm được của lần chạy đó, ta cần maximize giá trị này
- total_weight: không gian đã bị chiếm trong sack khi sử dụng kết quả này
- num_items: số item có thể chọn để bỏ vào sack
- num_packed: số item đã được chọn đẻ bỏ vào sack
- is_optimal: kết quả này có tối ưu không
- time_sec: thời gian chạy để ra được kết quả đó

  Solver giải bài knapsack ở trong bài tập này là: KNAPSACK_MULTIDIMENSION_BRANCH_A

  is_optimal được tìm ra bằng cách thời gian chạy thực tế có giống thời gian bị timeout không. Nếu giống thì có nghĩa là thuật toán chưa tìm ra được kết quả tối ưu. Nếu như thuật toán chạy xong trước khi bị timeout sẽ là tối ưu

1.2 Code và giải thích ý tưởng

1.2.1 Đọc file .kp

Các file này sẽ có format như sau: dòng đầu tiên là số item, dòng thứ 2 là weight tối đa, các dòng tiếp theo là value và weight của các item đó

Function sau sẽ đọc các file theo thứ tự đó và trả về value(profit), weight, và capacity(weight tối đa có thể chứa của sack)

```
def parse_knapsack_file(filepath):
    with open(filepath, 'r') as f:
        lines = [line.strip() for line in f if line.strip()]
        try:
            n = int(lines[0])
        except Exception as e:
            raise ValueError(f"Error parsing number of items in
             ↪  {filepath}: {e}")
        try:
            capacity = int(lines[1])
        except Exception as e:
            raise ValueError(f"Error parsing capacity in {filepath}:
             ↪  {e}")

        profits = []
        weight_list = []
        for line in lines[2:]:
            try:
                p, w = map(int, line.split())
            except Exception as e:
                raise ValueError(f"Error parsing profit and weight from
                 ↪  line '{line}' in {filepath}: {e}")
            profits.append(p)
            weight_list.append(w)
        # For a single-dimension knapsack, weights must be a list
         ↪  containing one list with all weights.
        weights = [weight_list]

    return profits, weights, [capacity]
```

1.2.2 Gọi solver

Phần code gọi solver này tham khảo từ phần code mẫu của OR tool google.

Ở đây sử dụng hàm set_time_limit cho solver để timeout sau 60 giây, tính thời gian và so sánh thử nó có gần với time timit không để biết là solver đã timeout hay đã tìm ra kết quả tối ưu

```
def solve_knapsack(profits, weights, capacities, time_limit_seconds=60):
    solver = knapsack_solver.KnapsackSolver(
```

```
        knapsack_solver.SolverType.KNAPSACK_MULTIDIMENSION_BRANCH_AND_BOUND_SOLVER,
        "KnapsackExample"
    )
    solver.set_time_limit(time_limit_seconds)
    solver.init(profits, weights, capacities)

    start_time = time.time()
    computed_value = solver.solve()
    duration = time.time() - start_time
    if abs(duration - time_limit_seconds) <= 0.05:
        optimal = "no"
    else:
        optimal = "yes"
    packed_items = [i for i in range(len(profits)) if solver.best_solution_contains(i)]
    total_weight = sum(weights[0][i] for i in packed_items)

    return {
        'value': computed_value,
        'weight': total_weight,
        'items': packed_items,
        'is_optimal': optimal,
        'duration': duration
    }
```

1.2.3 Hàm chọn test random và chạy code

Trong function này, ta sẽ chọn random số item từ 100, 200 và 500. Sau đó sẽ parse dir và gọi 2 hàm trên để parse file và chạy solver

Sau đó lưu các thông tin ở trên vào một list và sau đó viết kết quả vào một file csv

```
def run_random_tests(kplib_root, result_path="knapsack_random_results.csv"):
    base_path = Path(kplib_root) / "kplib"
    target_sizes = [100, 200, 500]
    target_size_dirs = {"n" + str(size).zfill(5) for size in target_sizes}
    results = []

    for group in sorted(base_path.iterdir()):
        if not group.is_dir():
            continue
        print(f"\nProcessing group {group.name}")

        valid_files = []
        for size_dir in target_size_dirs:
            size_path = group / size_dir
```

```
            if not size_path.exists() or not size_path.is_dir():
                continue
            # Traverse each capacity folder (e.g., R01000, R10000, etc.)
            for capacity_folder in size_path.iterdir():
                if not capacity_folder.is_dir() or not
                ↪ capacity_folder.name.startswith("R"):
                    continue
                # Collect all .kp files in the capacity folder.
                files = list(capacity_folder.glob("*.kp"))
                valid_files.extend(files)

        if len(valid_files) < 5:
            print(f"Group {group.name}: found only {len(valid_files)}
            ↪ valid test cases. Skipping group.")
            continue

        selected_tests = random.sample(valid_files, 5)
        for kp_file in selected_tests:
            try:
                print(f"Doing test file {kp_file}")
                relative_path = kp_file.relative_to(base_path)
                print(f"   Solving {relative_path}...", end=' ')
                profits, weights, capacities =
                ↪ parse_knapsack_file(kp_file)
                result = solve_knapsack(profits, weights, capacities)

                results.append({
                    'group': group.name,
                    'file': str(relative_path),
                    'value': result['value'],
                    'total_weight': result['weight'],
                    'num_items': len(profits),
                    'num_packed': len(result['items']),
                    'is_optimal': result['is_optimal'],
                    'time_sec': round(result['duration'], 2)
                })
                print(f"Done in {result['duration']:.2f}s | Optimal:
                ↪ {result['is_optimal']}")
            except Exception as e:
                print(f"Error solving {relative_path}: {e}")

    # Save results to CSV
    df = pd.DataFrame(results)
    df.to_csv(result_path, index=False)
    print(f"\n All results saved to: {result_path}")
```

Chương 2

Thống kê kết quả và nhận xét

2.1 Bảng kết quả

group	file	value	total_weight	num_items	num_packed	is_optimal	time_sec
00Uncorrelated	00Uncorrelated/n00100/R01000/s047.kp	40984	24919	100	62	yes	0.0
00Uncorrelated	00Uncorrelated/n00100/R10000/s085.kp	414484	233309	100	66	yes	0.0
00Uncorrelated	00Uncorrelated/n00500/R10000/s096.kp	1909483	1231288	500	307	yes	0.0
00Uncorrelated	00Uncorrelated/n00200/R01000/s086.kp	79960	50562	200	126	yes	0.0
00Uncorrelated	00Uncorrelated/n00100/R01000/s060.kp	39257	22713	100	61	yes	0.0
01WeaklyCorrelated	01WeaklyCorrelated/n00200/R01000/s009.kp	51864	46603	200	103	yes	0.0
01WeaklyCorrelated	01WeaklyCorrelated/n00100/R10000/s023.kp	272389	250091	100	50	yes	0.0
01WeaklyCorrelated	01WeaklyCorrelated/n00200/R10000/s093.kp	548932	495259	200	106	yes	0.0
01WeaklyCorrelated	01WeaklyCorrelated/n00100/R10000/s020.kp	244862	219438	100	53	yes	0.0
01WeaklyCorrelated	01WeaklyCorrelated/n00500/R01000/s037.kp	138374	125080	500	260	yes	0.0
02StronglyCorrelated	02StronglyCorrelated/n00100/R10000/s012.kp	299639	227639	100	72	yes	0.38
02StronglyCorrelated	02StronglyCorrelated/n00100/R10000/s023.kp	320095	250095	100	70	yes	18.2
02StronglyCorrelated	02StronglyCorrelated/n00500/R10000/s090.kp	1581816	1229816	500	352	no	60.0
02StronglyCorrelated	02StronglyCorrelated/n00500/R10000/s089.kp	1604205	1253205	500	351	no	60.0
02StronglyCorrelated	02StronglyCorrelated/n00500/R01000/s027.kp	163206	128606	500	346	no	60.0
03InverseStronglyCorrelated	03InverseStronglyCorrelated/n00100/R01000/s044.kp	24564	27664	100	31	yes	30.89
03InverseStronglyCorrelated	03InverseStronglyCorrelated/n00200/R01000/s092.kp	55576	62076	200	65	no	60.0
03InverseStronglyCorrelated	03InverseStronglyCorrelated/n00100/R01000/s054.kp	27127	30227	100	31	yes	0.44
03InverseStronglyCorrelated	03InverseStronglyCorrelated/n00500/R01000/s016.kp	127688	142788	500	151	no	60.0
03InverseStronglyCorrelated	03InverseStronglyCorrelated/n00500/R01000/s068.kp	133217	148917	500	157	no	60.0
04AlmostStronglyCorrelated	04AlmostStronglyCorrelated/n00200/R01000/s069.kp	63119	49114	200	140	no	60.0
04AlmostStronglyCorrelated	04AlmostStronglyCorrelated/n00200/R01000/s064.kp	62205	48005	200	142	no	60.0
04AlmostStronglyCorrelated	04AlmostStronglyCorrelated/n00100/R01000/s058.kp	31718	24819	100	69	no	60.0
04AlmostStronglyCorrelated	04AlmostStronglyCorrelated/n00200/R01000/s043.kp	64215	50333	200	139	no	60.0
04AlmostStronglyCorrelated	04AlmostStronglyCorrelated/n00100/R10000/s019.kp	296791	225775	100	71	yes	0.05
05SubsetSum	05SubsetSum/n00100/R10000/s073.kp	237227	237227	100	51	yes	0.0
05SubsetSum	05SubsetSum/n00200/R10000/s004.kp	493884	493884	200	104	yes	0.0
05SubsetSum	05SubsetSum/n00100/R01000/s042.kp	23764	23764	100	51	yes	0.0
05SubsetSum	05SubsetSum/n00200/R10000/s089.kp	494814	494814	200	97	yes	0.0
05SubsetSum	05SubsetSum/n00100/R10000/s075.kp	257206	257206	100	55	yes	0.0
06UncorrelatedWithSimilarWeights	06UncorrelatedWithSimilarWeights/n00100/R10000/s036.kp	37038	4902612	100	49	yes	0.02
06UncorrelatedWithSimilarWeights	06UncorrelatedWithSimilarWeights/n00100/R01000/s057.kp	36491	4902036	100	49	yes	0.15
06UncorrelatedWithSimilarWeights	06UncorrelatedWithSimilarWeights/n00500/R10000/s015.kp	193164	24712485	500	247	no	60.0
06UncorrelatedWithSimilarWeights	06UncorrelatedWithSimilarWeights/n00100/R01000/s016.kp	34448	4902554	100	49	yes	0.09
06UncorrelatedWithSimilarWeights	06UncorrelatedWithSimilarWeights/n00200/R10000/s079.kp	72893	9904794	200	99	yes	0.0
07SpannerUncorrelated	07SpannerUncorrelated/n00200/R10000/s086.kp	257402	272438	200	66	no	60.0
07SpannerUncorrelated	07SpannerUncorrelated/n00200/R10000/s074.kp	474067	62521	200	95	no	60.0
07SpannerUncorrelated	07SpannerUncorrelated/n00100/R01000/s014.kp	14616	19720	100	41	no	60.0
07SpannerUncorrelated	07SpannerUncorrelated/n00100/R10000/s044.kp	166738	119134	100	68	no	60.03
07SpannerUncorrelated	07SpannerUncorrelated/n00200/R10000/s038.kp	356510	214060	200	147	no	60.0
08SpannerWeaklyCorrelated	08SpannerWeaklyCorrelated/n00100/R10000/s099.kp	101479	76526	100	62	no	60.0
08SpannerWeaklyCorrelated	08SpannerWeaklyCorrelated/n00500/R01000/s088.kp	147957	36708	500	319	no	60.0
08SpannerWeaklyCorrelated	08SpannerWeaklyCorrelated/n00500/R01000/s047.kp	89892	49470	500	264	no	60.0
08SpannerWeaklyCorrelated	08SpannerWeaklyCorrelated/n00100/R01000/s012.kp	20568	12719	100	59	no	60.0
08SpannerWeaklyCorrelated	08SpannerWeaklyCorrelated/n00500/R01000/s034.kp	185746	71327	500	257	yes	59.38
09SpannerStronglyCorrelated	09SpannerStronglyCorrelated/n00200/R10000/s075.kp	757342	215342	200	101	no	60.0
09SpannerStronglyCorrelated	09SpannerStronglyCorrelated/n00100/R10000/s039.kp	372530	65530	100	61	no	60.0
09SpannerStronglyCorrelated	09SpannerStronglyCorrelated/n00200/R10000/s016.kp	798712	199712	200	113	yes	60.05
09SpannerStronglyCorrelated	09SpannerStronglyCorrelated/n00100/R10000/s058.kp	417829	92829	100	63	no	60.0
09SpannerStronglyCorrelated	09SpannerStronglyCorrelated/n00500/R10000/s037.kp	2449482	455482	500	366	yes	59.43
10MultipleStronglyCorrelated	10MultipleStronglyCorrelated/n00200/R01000/s070.kp	80212	49212	200	140	yes	0.04
10MultipleStronglyCorrelated	10MultipleStronglyCorrelated/n00500/R10000/s007.kp	1981090	1188090	500	351	no	60.0
10MultipleStronglyCorrelated	10MultipleStronglyCorrelated/n00500/R10000/s085.kp	1978402	1209402	500	350	no	60.0
10MultipleStronglyCorrelated	10MultipleStronglyCorrelated/n00100/R01000/s037.kp	39728	24228	100	69	yes	0.02
10MultipleStronglyCorrelated	10MultipleStronglyCorrelated/n00100/R10000/s007.kp	389831	232831	100	70	yes	0.0
11ProfitCeiling	11ProfitCeiling/n00200/R10000/s052.kp	505203	505222	200	83	no	60.0
11ProfitCeiling	11ProfitCeiling/n00200/R01000/s034.kp	52860	52878	200	91	yes	0.02
11ProfitCeiling	11ProfitCeiling/n00500/R01000/s020.kp	118389	118448	500	209	no	60.0
11ProfitCeiling	11ProfitCeiling/n00100/R01000/s091.kp	27042	27054	100	44	yes	0.01
11ProfitCeiling	11ProfitCeiling/n00200/R10000/s048.kp	477954	477965	200	79	yes	0.62
12Circle	12Circle/n00100/R01000/s050.kp	519375	24649	100	36	yes	0.0
12Circle	12Circle/n00200/R01000/s060.kp	993306	47141	200	73	yes	0.03
12Circle	12Circle/n00100/R10000/s017.kp	17728703	265944	100	38	yes	1.01
12Circle	12Circle/n00100/R10000/s021.kp	16964007	254473	100	37	no	60.0
12Circle	12Circle/n00200/R10000/s062.kp	32860537	492933	200	76	no	60.0

Bảng 2.1: Paired t-test of most common TF families for Pearson Correlations

2.2 Nhận xét

2.2.1 00 Uncorrelated Group

Vì dữ liệu không tương quan(weight và value được random) nên sẽ có các item weight thấp nhưng value cao và ngược lại nên dễ prune hơn nên dễ giải hơn

2.2.2 01 Weakly Correlated

Khi dữ liệu đã có sự tương quan nhưng tương quan yếu và số item không cao nên kết quả sẽ gần giống như dữ liệu không tương quan

2.2.3 02 strongly correlated

Dữ liệu có sự tương quan cao(value cao thì sẽ có weight cao) nên sẽ khó chọn item nào để bỏ vào sack => khó prune hơn nên giải sẽ khó hơn, các test đều timeout, kể cả test size nhỏ(100 items) cũng chạy mất 18 giây

2.2.4 03 Inverse strongly correlated

Các value cao sẽ có weight thấp và ngược lại.

Branch and bound sẽ khó khăn trong việc tìm được tổ hợp phù hợp của các item có weight nhỏ và chọn một số item có weight lớn để fill vào sack. Nên vấn đề này khá khó, đều timeout hoặc chạy lâu.

2.2.5 04 Almost strongly correlated

Giống với strongly correlated, vấn đề này khá khó và gần như các test đèu timeout

2.2.6 05 Subset Sum

Subset Sum data là data có weight = value, vì bài toán giờ chỉ là chọn sao để weight đạt được là cao nhất nên pruning sẽ dễ hơn và datasize trong test này chỉ random ra size 100 với 200 nên branch and bound solver đều chạy các test rất nhanh.

2.2.7 06 UncorrelatedWithSimilarWeights

Các test ở phần này chạy khá nhanh khi size nhỏ, nhưng khi size lớn hơn thì giải khó hơn và time out, vì các weight nó gần nhau nên khi size lớn lên thì có rất nhiều tổ hợp gần giống nhau, nên chạy sẽ khó

2.2.8 07 Spanner Uncorrelated

Test ở phần này rất khó, mặc dù size chỉ 100 và 200 nhưng đều không tìm ra kết quả và timeout. Điều này bởi vì Spanner Uncorrelated sẽ có nhiều item giống nhau với giá trị gần giống nên khó prune, do đó thời gian chạy sẽ cao.

2.2.9 08 Spanner Weakly Correlated

Khi base set tương quan yếu, lúc sinh ra test sẽ có nhiều giá trị gần giống hơn spanner uncorrenlated nữa, làm càng khó prune => tất cả các test đều timeout

2.2.10 09 Spanner strongly correlated

Giống với Spanner Weakly Correlated, Spanner strongly Correlated còn có càng nhiều giá trị giống nhau, càng khó prune. Ở đây có một bug là thời gian chạy sai khác thời gian đo được nhiều hơn 0.05 nên sẽ detect ra được là solution optimal nhưng thực ra là không optimal(Vì đã chạy hết thời gian và timeout nhưng chưa tìm được kết quả).

2.2.11 10 Multiple Strongly Correlated

Data sẽ được chia ra nhiều group, mỗi group sẽ tương quan cao, những test này chạy sẽ khó khi các group có giá trị xa nhau, dễ prune, nhưng nếu các group này có giá trị gần nhau thì sẽ khó giải hơn, làm cho khó prune => timeout

2.2.12 11 Profit Celling

Data sẽ có dạng: các item có weight <= C sẽ có profit bằng weight nhưng các item có weight > C thì profit sẽ bị cap ở C. Vì phân phối data này bị cap nên sẽ không phù hợp với sử dụng heuristic ratio trong branch and bound, do đó có một số test bị flat value nhiều quá sẽ thực hiện không tốt

2.2.13 12 Circle

Data Circle được design một cách đặc biết để lợi dụng lỗ hổng của greedy heuristic, branch and bound nên các test thường sẽ không giải đc và timeout nhưng mà trong phần này random ra được 3 test có size nhỏ nên vẫn giải tốt.

2.2.14 Kết luận

Spanner strongly correlated là khó nhất và Uncorrelated group là dễ nhất.