Benchmark Set Reduction for Cheap Empirical Algorithmic Studies

Mustafa MISIR
Istinye University, Department of Computer Engineering

mustafa.misir@istinye.edu.tr

http://mustafamisir.github.io
http://memoryrlab.github.io

IEEE CEC 2021
Outline

▶ Goal
▶ Method
▶ Traveling Thief Problem
▶ Computational Results
▶ Conclusion and Future Research
Algorithm Design is an iterative process in a loop of

- Implement $\leadsto$ Experiment $\leadsto$ Modify $\leadsto$ Experiment . . .
- Recurring experimentation can be a burden for whom with limited computational resources

---

2 image source: https://www.coursera.org/courses?query=datastructuresandalgorithms
Goal

Reducing a given benchmark set so that the experimental evaluation cost for the algorithmic studies can be significantly degraded

- specifically for large benchmark sets
Outline

- Goal
- Method
- Traveling Thief Problem
- Computational Results
- Conclusion and Future Research
Method | Inspiration – ALORS

An **Algorithm Selection / Recommender** system, operates through mapping instances’ features to instances’ **latent** (hidden) features

- Use **matrix factorization** to extract latent features – **Singular Value Decomposition (SVD)** is used

---

Method

$\mathcal{M}_{n \times m}$ is a rank matrix of $n$ instances and $m$ algorithms

Input
Performance matrix $\mathcal{M}_{n \times m}$
Matrix rank for dimensionality reduction $r$

Latent feature extraction
Apply SVD to $\mathcal{M}$ to extract $U_r$ and $V_r$

Instance clustering
Find best $k$ for $k$-means($U_r$) w.r.t. Silhouette score
Compute clusters $C_k$

Instance subset selection
Return $I_s = \bigcup \{ \text{select}(C_j, \left[ \frac{\text{size}(C_j)}{\min_{i=1 \ldots k} \text{size}(C_i)} \right]) \}$ for $j = 1 \ldots k$
Method | Feature Extraction (Step 1)

SVD is used to decompose $\mathcal{M}$:

$$\mathcal{M} = U\Sigma V^t$$

- $U$ is a matrix representing the rows of $\mathcal{M}$, i.e. instances
- $V$ is a matrix representing the columns of $\mathcal{M}$, i.e. algorithms
- $\Sigma$ is a diagonal matrix of sorted singular values, denotes importance

The dimensions of the resulting matrices can be reduced to $r$ by using the first $r$ dimensions

$$\mathcal{M} \approx U_r\Sigma_r V_r^t$$
Method | Instance Clustering (Step 2)

Explore different instance types through clustering

- $k$-means is used to cluster the instances based on the extracted latent instance features
- Silhouette score, i.e. mean Silhouette coefficient, is employed to evaluate cluster quality, for $k = \{2, \ldots, 100\}$
- Next, binary search is performed between $k = 101$ and $n/2$

---

7 figure source: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
The closest instances to the centroid of each cluster are selected, taking the cluster sizes into account

\[
\left\lceil \frac{\text{size}(C_j)}{\min_{i=1...k} \text{size}(C_i)} \right\rceil
\]

where \( \text{size}(C_j) \) is the number of instances in the cluster \( C_j \).
Outline

- Goal
- Method
- Traveling Thief Problem
- Computational Results
- Conclusion and Future Research
Traveling Thief Problem (TTP)\textsuperscript{11}

An \textbf{NP-hard} problem concerned with two other, well-known optimization problems, namely\textsuperscript{8}

\begin{itemize}
  \item Traveling Salesman Problem (TSP)\textsuperscript{9}
  \item Knapsack Problem (KP)\textsuperscript{10}
\end{itemize}

\textsuperscript{8} image source: \url{https://en.wikipedia.org/wiki/Knapsack_problem~/Travelling_salesman_problem}


A set of cities: \( N = \{1, \ldots n\} \)

A set of items: \( M = \{1, \ldots m\} \)

The distance between the city \( i \) and city \( j \): \( d_{ij} \)

The city \( i \) (except the starting city) has a set of items: \( M_i = \{1, \ldots, m_i\} \), \( M = \bigcup_{i \in N} M_i \)

The item \( k \) from the city \( i \) has profit \( p_{ik} \) and weight \( w_{ik} \)

\( W \) is the knapsack capacity

\( R \) is the renting rate (cost) for the knapsack, per time unit

\( v_{max} \) and \( v_{min} \) denote max and min speed of the thief, affected by the total weight of the collected items

---

The goal is to specify a tour maximizing the total profit

- The tour consists of all the cities exactly once, starting from the first city and returning back there

The objective function\(^{13}\) for a tour \(\Pi = (x_1, \ldots, x_n), x_i \in N\) and a packing plan \(P = (y_{21}, \ldots, y_{nm})\):

\[
Z(\Pi, P) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R \left( \sum_{i=1}^{n-1} \frac{d_{xi x_{i+1}}}{\nu_{max} - \nu W_{x_i}} \right) + \frac{d_{xn x_1}}{\nu_{max} - \nu W_{x_n}}
\]

where

- \(y_{ik} \in \{0, 1\}\) shows whether the item \(k\) is picked from the city \(i\)
- \(W_i\) is the total item weight when the thief leaves the city \(i\)
- \(\nu = \frac{\nu_{max} - \nu_{min}}{W}\) is a constant

\(^{13}\) Within the knapsack’s rent term, the first part is the traveling cost between cities while the second part refers to the cost of going back to the starting city.
9720 TTP instances, from the literature

- Based on TSPLIB\(^{14}\)
- Considering 3 KP variations, i.e. uncorrelated, uncorrelated with similar weights and bounded strongly correlated
- Different number of per city items
- Distinct renting rates, \( R \)

\(^{14}\) http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
21 candidate TTP algorithms, from the literature

- Simple Heuristic, Random Local Search (RLS), (1+1)-Evolutionary Algorithm (EA)
- Density-based Heuristic (DH)
- Memetic Algorithm with the Two-stage Local Search (MATLS)
- S1, S2, S3, S4, S5, C1, C2, C3, C4, C5, C6
- CoSolver with 2-OPT and Simulated Annealing (CS2SA)
- Variants of MAX-MIN Ant System (MMAS): MMASIs3 (M3), MMASIs4 (M4), MMASIs3boost (M3B), MMASIs4boost (M4B)
Outline

- Goal
- Method
- Traveling Thief Problem
- Computational Results
- Conclusion and Future Research
Spearman’s rank coefficient test evaluates the marginal algorithm contribution to any algorithm (portfolio) subset, for Oracle

- ranking is preserved in most cases, i.e. $\rho$-values of $> 0.9$
- Subset-k5-1 achieves with its 62 instances $\rho = 0.974$, which is the best score among the smallest subsets
- Overall, Subset-k5-20 achieves with its 1240 instances $\rho = 0.991$

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset-k5-1</td>
<td>0.974</td>
</tr>
<tr>
<td>Subset-k5-5</td>
<td>0.986</td>
</tr>
<tr>
<td>Subset-k5-10</td>
<td>0.988</td>
</tr>
<tr>
<td>Subset-k5-20</td>
<td>0.991</td>
</tr>
<tr>
<td>Subset-k5-30</td>
<td>0.988</td>
</tr>
<tr>
<td>Subset-k6-1</td>
<td>0.805</td>
</tr>
<tr>
<td>Subset-k6-5</td>
<td>0.813</td>
</tr>
<tr>
<td>Subset-k6-10</td>
<td>0.788</td>
</tr>
<tr>
<td>Subset-k6-20</td>
<td>0.804</td>
</tr>
<tr>
<td>Subset-k6-30</td>
<td>0.808</td>
</tr>
<tr>
<td>Subset-k7-1</td>
<td>0.957</td>
</tr>
<tr>
<td>Subset-k7-5</td>
<td>0.970</td>
</tr>
<tr>
<td>Subset-k7-10</td>
<td>0.979</td>
</tr>
<tr>
<td>Subset-k7-20</td>
<td>0.986</td>
</tr>
<tr>
<td>Subset-k7-30</td>
<td>0.986</td>
</tr>
<tr>
<td>Subset-k8-1</td>
<td>0.971</td>
</tr>
<tr>
<td>Subset-k8-5</td>
<td>0.981</td>
</tr>
<tr>
<td>Subset-k8-10</td>
<td>0.981</td>
</tr>
<tr>
<td>Subset-k8-20</td>
<td>0.983</td>
</tr>
<tr>
<td>Subset-k8-30</td>
<td>0.979</td>
</tr>
</tbody>
</table>
62 selected TTP instances as a representative benchmark set for the complete 9720 instances
Computational Results | Feature Importance

For 55 TTP features, from the literature
Outline

- Goal
- Method
- Traveling Thief Problem
- Computational Results
- Conclusion and Future Research
Conclusion and Future Research

The proposed method is able to come up with representative instance sets, with less than %1 of the complete instance set.

Follow-up research:

▶ repeating the analysis on other problems
▶ offering a new clustering approach for determining the number of clusters cheaper
▶ benefiting from Matrix Completion (MC)\textsuperscript{15,16} to expand the applicability of the method
▶ recommending instance subsets not as a representative set of the large one but small yet a fair benchmark set

\textsuperscript{15} M. Mısır. Data sampling through collaborative filtering for algorithm selection, in the 16th IEEE CEC, 2017, pp. 2494–2501