BIGO 模型下禁忌搜索项目分组初始化的优化

孙智聪
计算机科学与工程系
南方科技大学
深圳，中国
12032471

摘要—带有分组约束的仓库货物存储位置问题 SLAP-GC 是典型的组合优化问题，该问题可以被分解为两个子问题，一是如何对产品 Product 内对 Item 分组，二是在确定分组的基础上如何对 Item 的位置进行分配。针对这个问题，论文 [1] 提出了一种双层优化模型 BIGO，并使用禁忌搜索算法寻找最优策略。但是该论文在禁忌搜索前的策略初始化上存在两个薄弱点，一是 Item 的初始分组采用的是随机分组，二确定分组方式后的初始分配策略则采用了贪心算法。针对以上问题，本文设计了 7 个优化算子，4 组实验。其中重点针对第一个薄弱点进行优化，即在初始分组过程中使用了挑拣频率的 3 种不同的启发信息，经 Wincoxon singned-rank test 统计，使用 K-Means 聚类的方法在 4 个数据集上都表现得更优，能够非常有效的得到更好的初始分组策略，有利于加快禁忌搜索的收敛速度，且该方法在其他论文中不曾见过。除此之外，本文还对 inslfSort 以及局部搜索 LNS 等进行优化测试，并对优化结果不明显的原因进行分析。

***Index Terms*—组合优化，SLAP-GC，BIGO，启发信息，聚类，K-Means**

I. 介绍

本文将在 Section I 中介绍分组约束下的仓储位置分配问题 SLAP-GC, 以及论文 [1] 中提出的双层优化模型 BIGO 和相应禁忌搜索 Tabu search 的算法。在 Section II 中，本文将讨论论文 [1] 中所提出算法存在的问题，并讨论解决思路、分析解决过程，并给出最终算法。在 Section III 中，本文将用可视化和统计两种方式比较论文 [1] 中算法和 7 种测试算法的性能，并给出相应分析；最后，Section IV 中将总结本文工作，并提出的解决目前算法存在问题的可能方案以及未来工作。

A. 分组约束下的仓储位置分配问题 SLAP-GC

图 1 为仓库布局示例，有 8 个货架 shelves，每个货架包含 5 个箱子 bins。白色区域表示走廊，灰色列表示垂直货架。区域 PD 表示挑拣/放置的起始位置。负责挑拣或者放置货物的工人需要从 PD 出发，前往某个 bin 挑拣或者放置一个 item，再回到 PD 点。而一个仓库中的货物包括不同产品，每个产品中又有不同的 item，比如产品 T-shirt 有 XL,L 两种 item。在这里假设仓库中的所有 item 数量和箱子 bin 的数量是相同的。

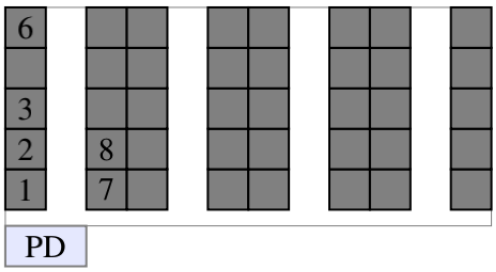


图 1. 仓库布局示例

仓库中典型的操作分为四类，收货 receiving、储存 storage、挑选 picking 和交付 delivering, 其中挑选 picking 是成本最高的工序。货物的不合理放置会影响到挑选时候的运营成本，因此需要考虑仓储位置分配问题，即 SLAP。SLAP-GC 的总体目标是在分组约束条件下，使得运营成本最小化，本文使用总挑选距离作为运营成本，目标函数如公式（1）所示：

$$\min \varsigma(x) = \sum_{i=1}^{N} \sum_{l=1}^{N} f_i \left(v_l + h_l\right) x_{il} \tag{1}$$

式（1）中 N 表示所有产品中 item 的数目，也是 bin 的数量，f_i 表示编号为 i(i=1,2,...,N) 的 item 的挑拣频率，v_l 表示挑拣该 item 需要移动的垂直距离，h_l 表示挑拣该 item 需要移动的水平距离，x_{il} 为 1 表示该 item 被放置在 l 位置，否则没有放置在 l 位置。

本文使用到的分组约束 Grouping Constrained 是该优化问题的最大挑战，它有两个限制。一是每个产品 product 只能被分为不超过两个的 item 组，二是每个组 gruop 必须被放置在临近位置。如图 1 所示，放置了 1 到 3 的 bin 是临近的，二而 6 和 7、8 不临近。第二个限制是最为苛刻的。

B. 用于 SLAP-GC 的双层优化模型 BIGO

针对 SLAP-GC 问题，论文 [1] 提出一种双层优化模型，上层优化分组，即 item grouping，下层优化确定分组下的分配，即 item assignment。其中，上层优化采用了下层优化的最优结果作为 fitness。在这个模型下，SLAP-GC 的目标转换为，遍历合适的可行分组，找到最优的分组 G^* 下的最优分配策略 $S^*(G^*)$，如式 (1) 所示：

$$\text{s.t.: } S \in \arg\min_{S' \in \Omega_S(G)} \phi(S') \ , \tag{2}$$

其中

$$\phi(S) = \sum_{i=1}^{M}\sum_{j=1}^{L}(i+j) f\left(e_{ij}(S)\right) \tag{3}$$

具体而言，针对这个模型，论文 [1] 中提出了两个搜索算法，其中性能最优的是禁忌搜索算法，算法的伪代码如图 2 所示。算法的整体思路是随机生成一个初始的分组，利用贪心算法生成在该分组方式下的初始 item 分配，接着评价初始策略的适应度，随后进行禁忌搜索。

Algorithm 4 Tabu search for SLAP-GC using the BIGO model

$S_0 \leftarrow null$;
while $S_0 = null$ **do**
  Randomly generate a grouping G;
  $S_0 \leftarrow \texttt{Init}(G)$;
end while
$(fit(G), S) \leftarrow \texttt{Evaluate}(G, S_0)$;
Initialize the tabu list *tabuList* $\leftarrow \emptyset$;
$G^* \leftarrow G$, $S^* \leftarrow S$;
while Stopping criteria are not met **do**
  $G_{next} \leftarrow null$, $S_{next} \leftarrow null$, $bestFitOneItr \leftarrow \infty$;
  for each $(G', S') \in \mathcal{N}(G, S)$ **do**
    $(fit(G'), S^*(G')) \leftarrow \texttt{Evaluate}(G', S')$;
    if G' is not tabu **or** G' satisfies the aspiration criterion **then**
      if $fit(G') < bestFitOneItr$ **then**
        $G_{next} \leftarrow G'$, $S_{next} \leftarrow S^*(G')$;
        $bestFitOneItr \leftarrow fit(G')$;
      end if
    end if
  end for
  $G \leftarrow G_{next}$, $S \leftarrow S_{next}$;
  if $fit(G) < fit(G^*)$ **then**
    $G^* \leftarrow G$, $S^* \leftarrow S$;
  end if
  Update *tabuList* according to G_{next}, and remove the tabu elements whose tabu duration have reached the tabu tenure;
end while
return (G^*, S^*);

图 2. 禁忌搜索算法伪代码图，源自论文 [1]

II. 优化思路与改进的算法

A. 优化思路

分析论文 [1] 的算法可以发现原算法有两个薄弱点：一是 Item 的初始分组采用的是随机分组，当初始分组与最优分组差距较大时，需要花费较长的时间才能收敛到最优分组附近；二确定分组方式后的初始分配策略则采用了贪心算法，将最大的组分配给最大的可行位置，该贪婪算法有时候会出现找不到可行解的问题，降低了效率，并且贪婪的结果不一定是最优的，从非最优到最优分组，又需要一定的收敛时间。

针对薄弱点一，需要对将分组问题拆分为两个子问题，即：

1) 需要将此 product 分为几个组?
2) 此 product 的每次组需要放置几个 items?

子问题 1 的答案是分为两个组，这个问题可以通过一个例子来解释，如图 3 和图 4 所示，假设一个 product 有 5 个 item，采用分组方式一即将 item 分为两组，采用分组方式二即将 item 分为一组两种情况下的目标函数即总挑选距离是一样的，但是分为两组的方式有一个优点，即局部搜索时操作的粒度更细，搜索空间更大，搜索到最优解的可能性更高。

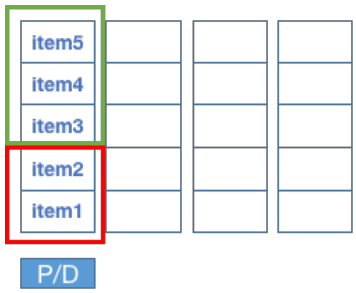


图 3. 分组方式一：两个分组

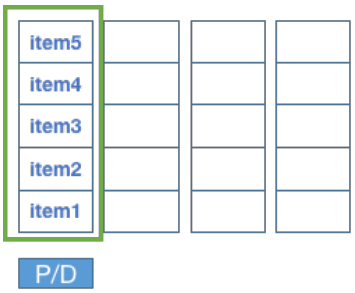


图 4. 分组方式二：一个分组

子问题 2 是在问题 1 的基础上，考虑两个分组需要各分配几个 item，原算法采用随机分配，最大的问题是没有使用启发信息，应利用启发信息将类似的 item 分在一组，item 的挑拣频率即是一个很好的启发信息，可以利用一个 product 中的挑拣频率的中位数、平均数等将 item 分为两组，更进一步考虑，每个 group 有几个 item 其实是一个聚类问题，可以通过 K-Means 聚类得到每个 group 的 item 数目。

B. 改进算法

针对以上思路，本文设计了 6 个改进算法，分别为：

1) A2: 基于挑拣频率均值的 item grouping
2) A3: 基于挑拣频率中位数的 item grouping
3) A4: 基于 K-Means 的 item grouping（方式一）
4) A5: 基于 K-Means 的 item grouping（方式二）
5) A6: 基于 K-Means 的 item grouping（方式一）+ 使用中位数的 inslfSort
6) A7: 基于 K-Means 的 item grouping（方式一）+ 改进的局部搜索

其中，A2 和 A3 算法统计每个 product 中 item 的挑拣频率的平均值（中位数），以其为分界线确定两组的 item 数目，比如一个 product 的 items 的挑拣频率为 1,3,10,40, 其均值为 13.5，中位数为 6.5。按照算法 A2，初始分组为：group1 $= \{1, 3, 10\}$，group2 $= \{40\}$。按照算法 A3, 初始分组为：group1 $= \{1, 3\}$，group2 $= \{10, 40\}$。

算法 A4 和 A5 都为 K 均值聚类算法，其中 A4 伪代码如下 K-Means1 所示。A4 和 A5 的区别在于：A4 的初始簇中心是在当前聚类 product 的所有 item 频率中随机选取的，而 A5 则对当前聚类 product 的所有 item 的频率进行排序，相对均匀地分为两大部分，接着两部分中各自随机生成一个簇中心，A5 的簇中心初始化方法考虑的是，如果在所有 item 频率中随机选取簇中心，很有可能出现极端的情况，比如两个簇中心的频率是 item 中最高的前两个，这样聚类的过程可能会花费更多的时间，因此 A5 的初始化方法相对而言避免了这种情况的出现。

K-Means 1: Use K-Means to item grouping

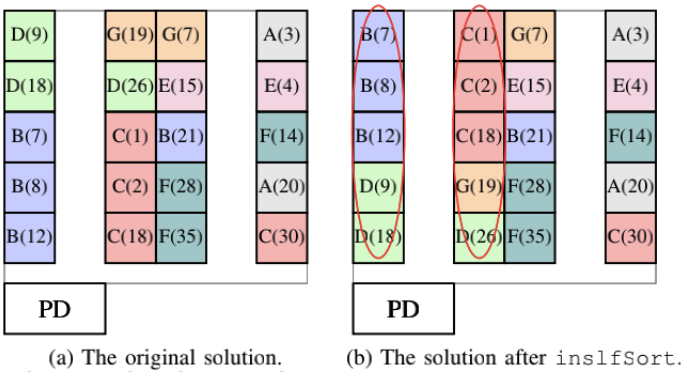
Input: Frequencies of items in an product $F = \{f_1, f_2, ..., f_n\}$, item i is represented by its frequency f_i
Cluster number K (K=2)

Output: Item grouping solution G;

Randomly select K items from F as the initial mean vectors $\{\mu_1, ..., \mu_K\}$;
do
  Let $C_i = \varnothing (1 \le i \le K)$
  for j = *1,2,...,n* **do**
    Calculate the distance between f_j and each mean vector $\mu_i (1 \le i \le K)$: $d_{ij} = ||x_j - \mu_i||_2$;
    The cluster label of f_j is determined by the nearest mean vector: $\lambda_j = argmin_{i \in \{1,...,k\}} d_{ij}$;
    Put the item f_j into the corresponding cluster: $C_{\lambda_j} = C_{\lambda_j} \cup \{x_j\}$;
  end
  for i = *1,...K* **do**
    Calculate the new mean vector:$\mu_i^{'} = \frac{1}{C_i} \sum_{x \in C_i} x$;
    if $\mu_i^{'} \neq \mu_i$ **then**
      Update current μ_i to $\mu_i^{'}$;
    end
    else
      Keep the current mean vector unchanged;
    end
  end
while *The centers of K clusters have not changed*;

算法 A6 是对确定分组后的初始 item 分配进行优

化。论文 [1] 中生成初始分组后，对分组先使用了局部搜索 LNS 以获得初始的分配方案，LNS 中使用到了 inslfSort、slfSort、itemSort、groupSort 四种排序算子，其中 inslfSort 的作用是将同一个 shelf 上的 group 进行排序和位置分配，使用的是 group 的频率均值。而算法 A6 则是使用频率的中位数进行排序和位置分配。



(a) The original solution. (b) The solution after `inslfSort`.

图 5. Shelf 内的 gruop 进行排序和位置分配，图源自论文 [1]

算法 A7 则是对论文开源算法的工程改进，论文 [1] 中局部搜索 LNS 使用一个条件循环来寻找最优的初始分配方案具体策略是：若当前尝试的分派方案对应的目标函数值比上一次尝试的值高，则跳出搜索，将上一次的分配方案输出为 item 分配的初始方案，供禁忌搜索使用。这个策略的缺点是跳出搜索的条件，因为当前尝试的分派方案对应的目标函数值比上一次尝试的值高，不代表搜索方向出错，可能再经过几轮的洗牌搜索可以获得比目前更好的值，常见的搜索算法如一致均衡搜索中这种例子很常见。

III. 实验设计与结果讨论

A. 实验设置

本文使用到的数据是论文 [1] 开源代码中的数据集 10-1、10-2、10-3 和 10-4，四个数据集都包含了 100 个 items，每个数据集中 product 包含的 item 数目统计图如图 6-图 9 所示，从统计图中可见大部分 product 包含的 item 数量都较少，大部分 product 只包含一个 item。对应的仓库大小设置为 5 个 shelves，每个 shelf 都有 5 个 bins。

根据论文 [1] 的建议，本文使用算法在小数据集上的运行时间都设定为 10 秒，禁忌搜索的 tenure coefficient 设置为 0.7。

本文使用到的统计测试方法为 **Wilcoxon signed-rank test**，假设的 H_0 两组数据的差异遵循围绕零的对称分布，即两组数据无显著差异，其中显著性水平都设置为 0.05。

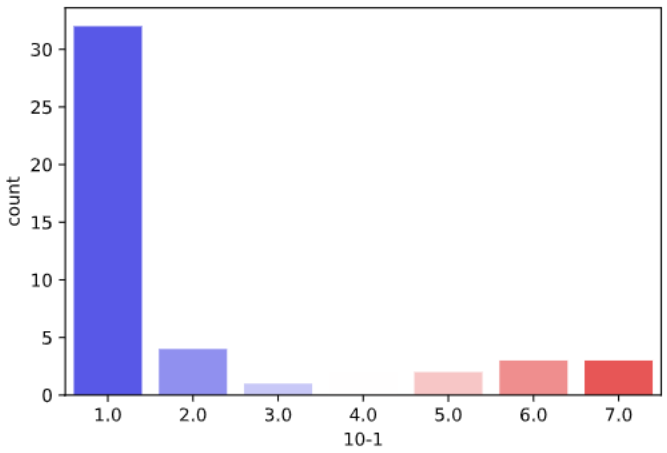


图 6. 数据集 10-1 中 product 包含的 item 数目统计图

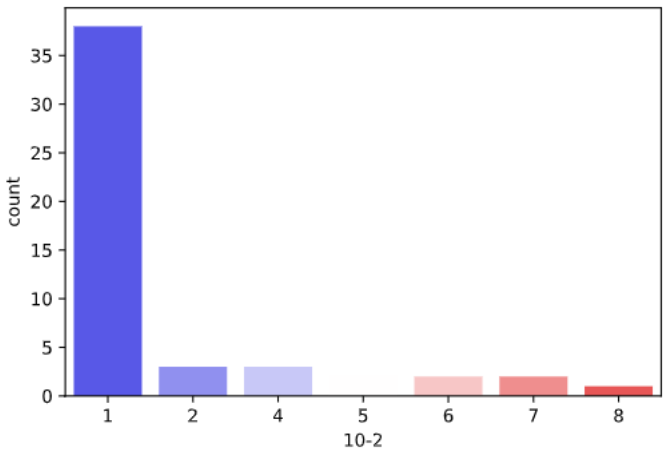


图 7. 数据集 10-2 中 product 包含的 item 数目统计图

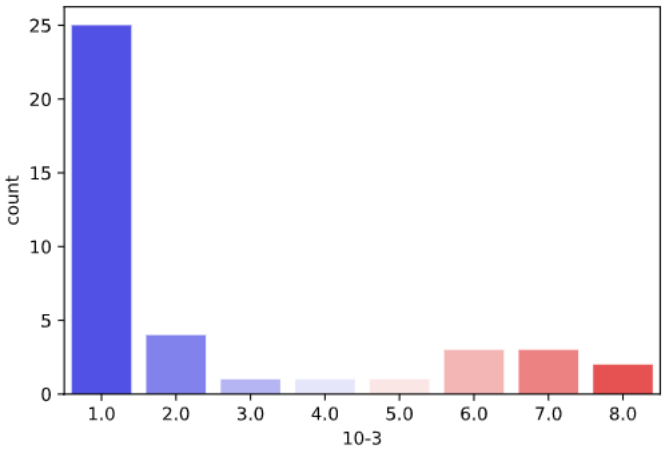


图 8. 数据集 10-3 中 product 包含的 item 数目统计图

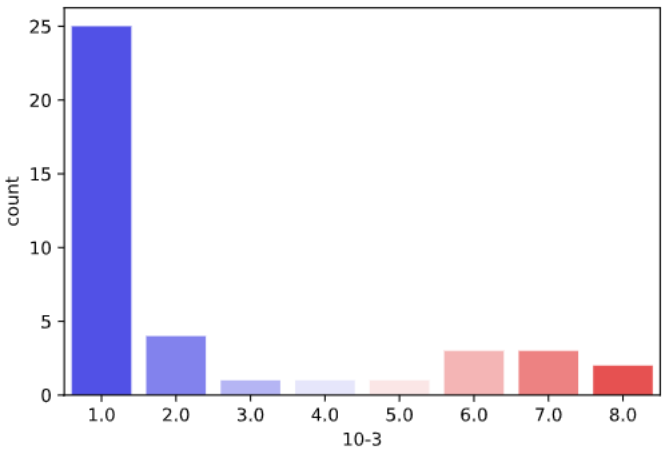


图 9. 数据集 10-3 中 product 包含的 item 数目统计图

B. 实验内容和分析

本文共仿真测试了 7 个算法，包括论文中的禁忌搜索算法（记为 A1），以及上文提到的 6 个优化算法。

实验使用到的算法均进行了 100 次的独立重复运行，并在 4 个数据集上统计了 100 次运行下初始 item 分配策略对应目标函数值的 **平均值、方差、最小值、秩和检验的总秩**，记录在表格中。表格中，左侧的第一列为对比的基准算法（如表格 1 中的 A1），Ranksum 表示 Wilcoxon signed-rank test 的总秩，值越大表示该算法最优值优于基准算法的程度越高。Win 的值表示该算法在此数据集上与基准算法的平均初始值的比较结果，Win = 1 表示优于基准算法（在本文中即值小于），Win = -1 表示差于基准算法，Win = 0 表示相同。在每个表格中，基准算法的 Ranksum 值、Win 值都置 Null 以作标识。

每组实验都使用一次具有代表性的运行数据，生成 **收敛曲线图**，曲线图的纵轴是 Best-so-far，横轴是迭代次数。

针对这些算法设计的 4 组实验，其中**优化效果最明显的是实验一**：

1) ***实验 1:*** 对比 item 挑拣频率的不同启发信息对初始分组效果的影响，实验包括算法 A1,A2,A3,A4。四个算法在四组数据的统计数据如表 1-表 4 所示，收敛曲线如图 10-图 13 所示。

10-1	**A1**	**A2**	**A3**	**A4**
Mean	35574.57	35581.25	35572.54	**35563.42**
Var	5100.5	288	**0.5**	3100.5
Min	35529	35762	35556	**35463**
Win	Null	-1	1	1
Ranksum	Null	-261	149	337

表 I

COMPARE 4 ALGORITHMS ON 10-1

10-2	**A1**	**A2**	**A3**	**A4**
Mean	17943.91	17936.54	17952.18	**17935.64**
Var	18818	3362	3612.5	**3612.5**
Min	**17863**	17964	17938	17921
Win	Null	1	-1	1
Ranksum	Null	332	-550	516

表 II

COMPARE 4 ALGORITHMS ON 10-2

10-3	**A1**	**A2**	**A3**	**A4**
Mean	38356.87	38349.87	38343.02	**38342.28**
Var	31250	363538	97240.5	**3120.5**
Min	37997	38132	38315	**37698**
Win	Null	1	1	1
Ranksum	Null	152	184	355

表 III

COMPARE 4 ALGORITHMS ON 10-3

10-4	**A1**	**A2**	**A3**	**A4**
Mean	17507.82	17501.83	17510.06	**17475.85**
Var	2380.5	21632	6160.5	**1860.5**
Min	**17389**	17396	17454	17401
Win	Null	1	-1	1
Ranksum	Null	787	537	1532

表 IV

COMPARE 4 ALGORITHMS ON 10-4

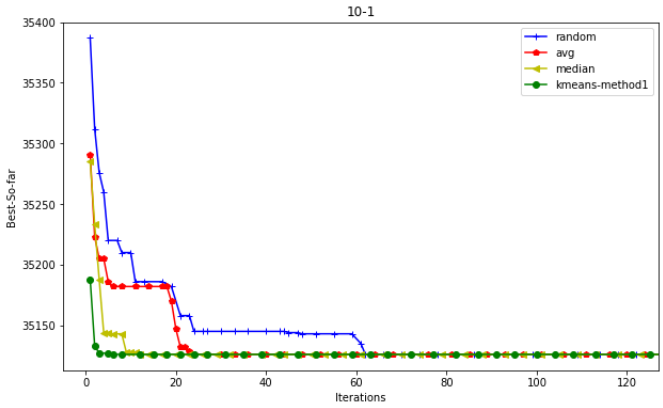


图 10. Compare Best-so-far on 10-1

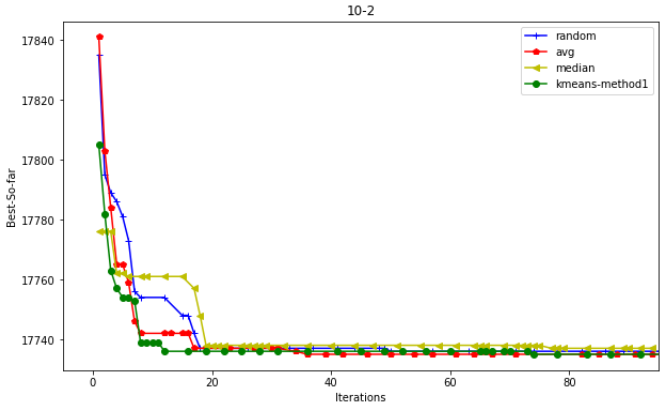


图 11. Compare Best-so-far on 10-2

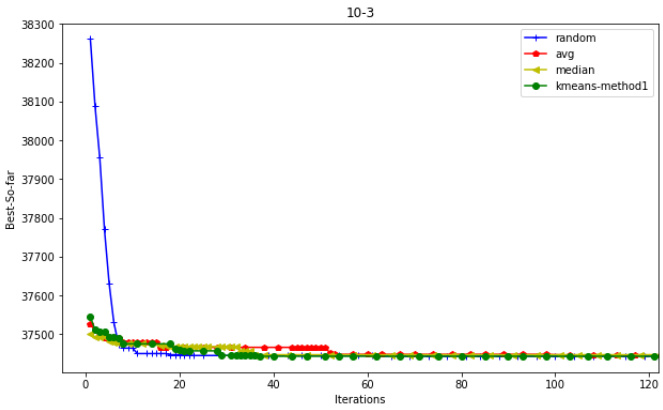


图 12. Compare Best-so-far on 10-3

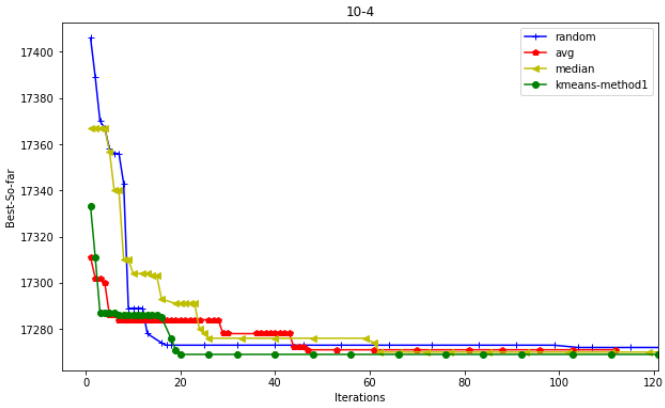


图 13. Compare Best-so-far on 10-4

讨论：（1）分析表 1-表 4 可知，通过 K-Means 聚类的算法 A4 得到的初始策略目标函数值的平均值在所有数据上都比原始算法更低，在方差和最低值上与原始算法不相上下，原始算法使用的随机策略搜索空间更大，因此在某些数据集上能够得到更低的初始策略目标函数值，但是不稳定。（2）分析图 10-图 13 可发现，A2,A3,A,A4 三个优化算法均有利于搜索更快地收敛，这反映了初始策略对收敛速度的影响，初始策略更接近最优策略，在同样的搜索方案下能够更快地找到最优策略。

以上两点分析表明，A2,A3,A4 三种优化算法确实可以得到更好的初始策略，并加快算法向最优策略收敛，而其中 K-Means 聚类的算法效果最好，在四个数据集上的表现都比论文 [1] 中的算法更优。

2) ***实验 2:*** 对比不同 K-Means 初始化方式对初始分组效果的影响，实验的算法为 A4 和 A5。两个算法在四组数据的统计数据如表 5-表 8 所示，收敛曲线如图 14-图 17 所示。

10-1	**A4**	**A5**
Mean	**35563.42**	35565.4
Var	3100.5	**2738**
Min	**35463**	35553
Win	Null	-1
Ranksum	Null	217

表 V
COMPARE 2 ALGORITHMS ON 10-1

10-2	A4	A5
Mean	**17935.64**	17945.42
Var	3612.5	**288**
Min	17921	**17928**
Win	Null	-1
Ranksum	Null	-537

表 VI

COMPARE 2 ALGORITHMS ON 10-2

10-3	A4	A5
Mean	38342.28	**38322.74**
Var	**3120.5**	136764.4
Min	**37698**	38016
Win	Null	1
Ranksum	Null	200

表 VII

COMPARE 2 ALGORITHMS ON 10-3

10-4	A4	A5
Mean	**17475.85**	17509.83
Var	**1860.5**	13448
Min	17401	**17363**
Win	Null	-1
Ranksum	Null	-1591

表 VIII

COMPARE 2 ALGORITHMS ON 10-4

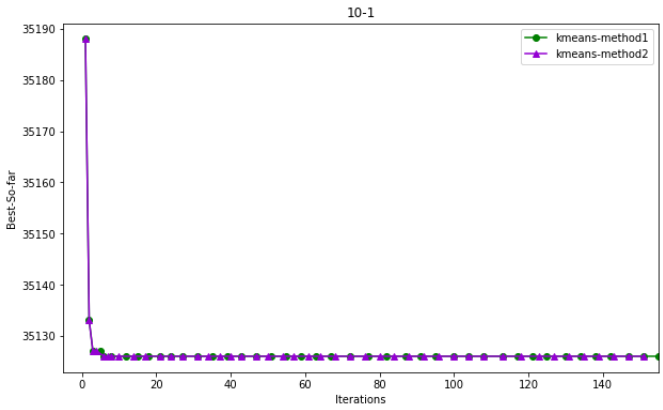


图 14. Compare Best-so-far on 10-1

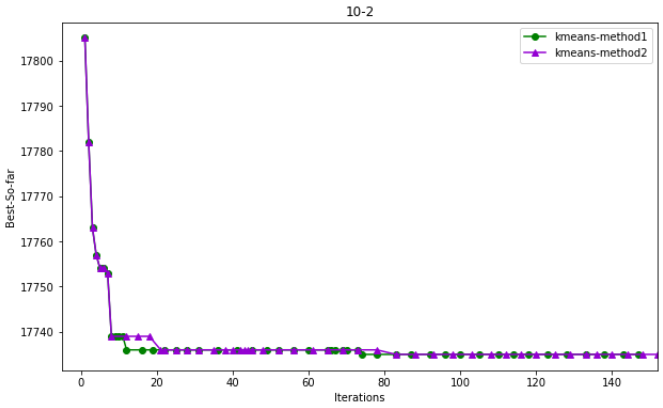


图 15. Compare Best-so-far on 10-2

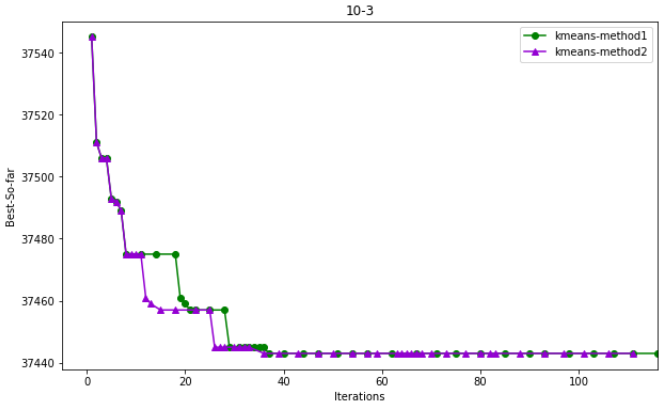


图 16. Compare Best-so-far on 10-3

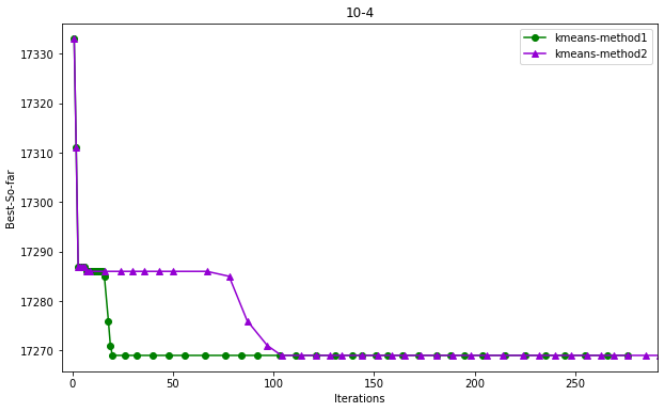


图 17. Compare Best-so-far on 10-4

讨论：分析表 5-表 8 以及图 14-图 17 可知，两种簇中心初始化方式下的初始 item 分配策略目标函数值相近，对收敛速度的优化效果相当。通过分析具体的数

据集，可发现数据集中每个 product 的 item 数目最多不超过 10，因此 A5 并没有表现出应有的优势的原因是，数据集较小，上文提到的极端情况（初始化簇中心对应的 item 频率都在数据的最高值端或最低值端）对算法的影响小到可以忽略。

3) ***实验 3:*** 对比使用 item 频率均值和中位数进行初始局部搜索 LNS，对初始分组效果的影响，实验对应的算法为 A4 和 A6。两个算法在四组数据的统计数据如表 9-表 12 所示，收敛曲线如图 18-图 21 所示。

10-1	A4	A6
Mean	**35563.42**	35571.03
Var	**3100.5**	39820
Min	35463	**35453**
Win	Null	-1
Ranksum	Null	-693

表 IX

COMPARE 2 ALGORITHMS ON 10-1

10-2	A4	A6
Mean	**17935.64**	17944.46
Var	**3612.5**	5304.5
Min	**17921**	17963
Win	Null	-1
Ranksum	Null	-421

表 X

COMPARE 2 ALGORITHMS ON 10-2

10-3	A4	A6
Mean	38342.28	**38337.45**
Var	3120.5	**2244.5**
Min	37698	**38308**
Win	Null	1
Ranksum	Null	86

表 XI

COMPARE 2 ALGORITHMS ON 10-3

10-4	A4	A6
Mean	17475.85	**17461.28**
Var	1860.5	**722**
Min	17401	**17436**
Win	Null	1
Ranksum	Null	376

表 XII

COMPARE 2 ALGORITHMS ON 10-4

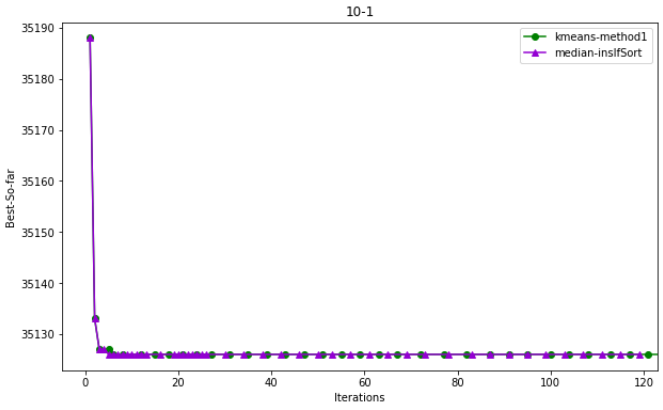


图 18. Compare Best-so-far on 10-1

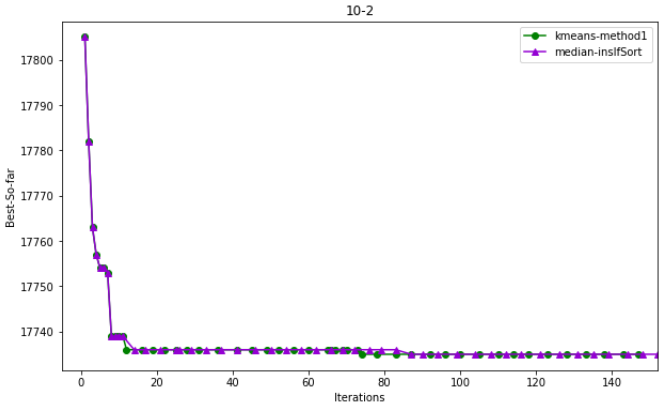


图 19. Compare Best-so-far on 10-2

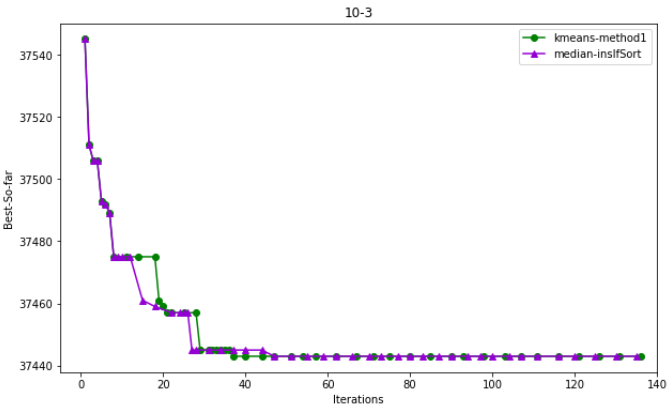


图 20. Compare Best-so-far on 10-3

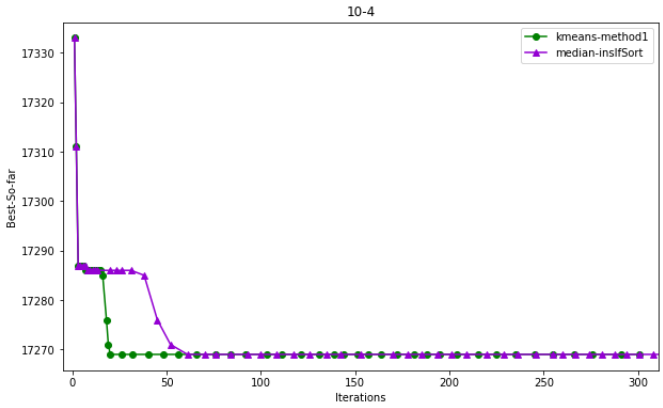


图 21. Compare Best-so-far on 10-4

讨论：分析表 9-表 12 以及图 18-图 21 可知，使用 item 频率中位数和均值进行 isfSort 的到的初始化分配策略的效果相当，并未进行很好的优化。原因可能是：item 频率平均数相比于中位数更具有代表性，通过均值能够真正将整体挑选距离降低，而因为偶然性原因中位数可能能够达到和平均数一样的效果，但是该效果不具有稳定性。

4) **实验 *4*:** 对比论文 [1] 使用的局部搜索 LNS 和本文提出的优化的初始局部搜索 LNS，对初始分组效果的影响，实验对应的算法为 A4 和 A7。两个算法在四组数据的统计数据如表 13-表 16 所示，收敛曲线如图 22-图 25 所示。

10-1	A4	A7
Mean	**35563.42**	35585.03
Var	**3100.5**	3528
Min	**35463**	35551
Win	Null	-1
Ranksum	Null	-48

表 XIII

Compare 2 algorithms on 10-1

10-2	A4	A7
Mean	**17935.64**	17944.25
Var	3612.5	**3528**
Min	**17921**	17923
Win	Null	-1
Ranksum	Null	-407

表 XIV

Compare 2 algorithms on 10-2

10-3	A4	A7
Mean	38342.28	**38307.24**
Var	**3120.5**	8844.5
Min	**37698**	38271
Win	Null	1
Ranksum	Null	460

表 XV

Compare 2 algorithms on 10-3

10-4	A4	A7
Mean	**17475.85**	17499.49
Var	**1860.5**	6384.8
Min	17401	**17439**
Win	Null	-1
Ranksum	Null	-1306

表 XVI

Compare 2 algorithms on 10-4

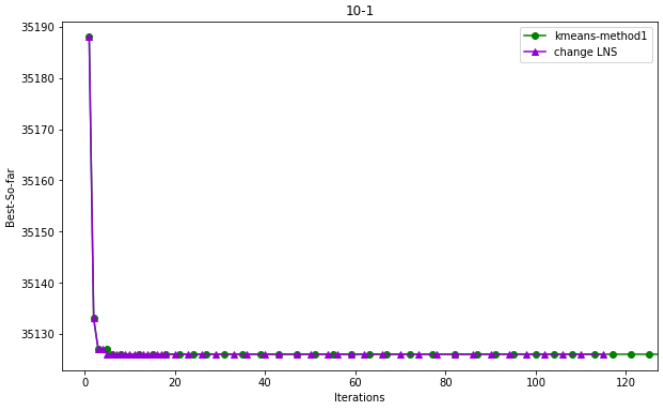


图 22. Compare Best-so-far on 10-1

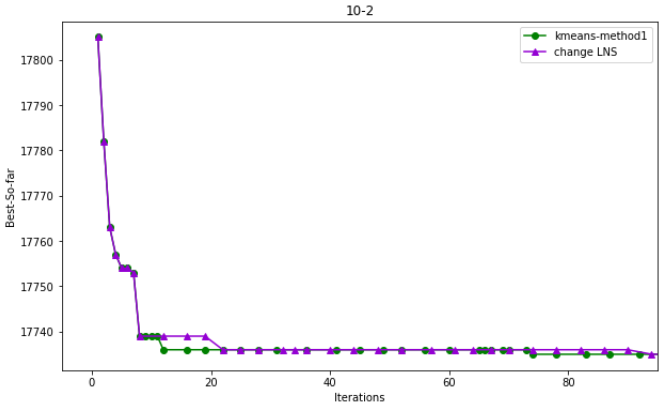


图 23. Compare Best-so-far on 10-2

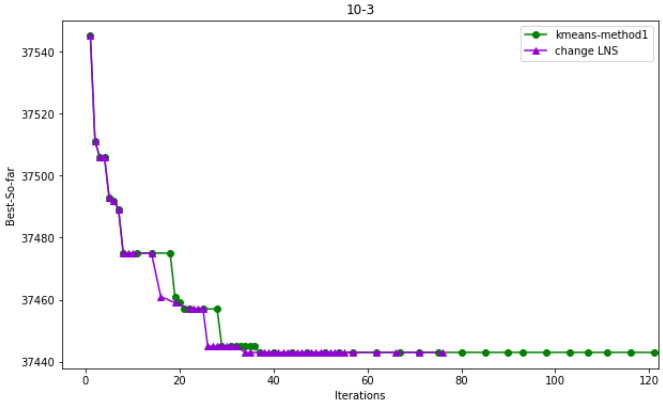


图 24. Compare Best-so-far on 10-3

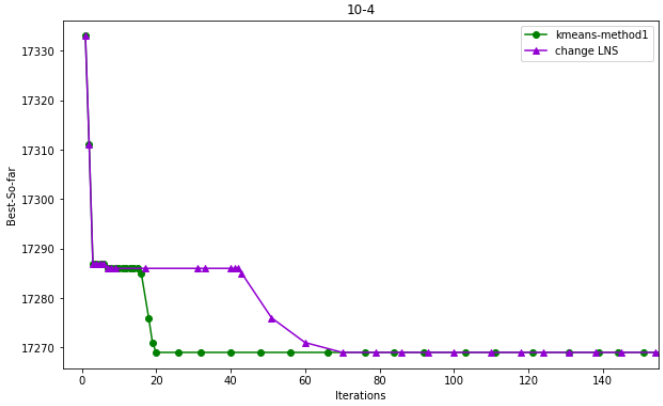


图 25. Compare Best-so-far on 10-4

讨论：分析表 13-表 16 以及图 22-图 25 可知，修改后的局部搜索 LNS 仅在数据集 3 上表现得比论文 [1] 的算法更优，但是在其他数据集上表现更差，优化效果不明显。原因是：在数据集小情况下，初始局部搜索在几次迭代就能够很快的收敛到局部最优，多余的迭代已经无法再提升初始策略，因此两者性能相当。

IV. 总结和未来的工作

本文首先介绍了 SLAP-GC 问题以及论文 [1] 提出的 BIGO 模型，接着分析 BIGO 模型下算法初始化的两个薄弱点，即 item 随机初始分组和 item 贪婪分配问题。本文仅针对第一个薄弱点进行优化，提出了将分组问题拆分为两个子问题，即分为几个组，每次组需要几个 item 两个小问题，并利用 item 的挑拣频率作为启发信息，设计了 4 组实验和 6 个优化算法。其中算法 A2-A4 是在论文 [1] 算法基础上的优化，A5-A7 是在算法 A4 的基础上尝试优化。经过收敛曲线和 Wilcoxon signed-rank test 的统计分析，本文提出的算法 A4-A7 在 4 个数据集上均达到比论文 [1] 中算法更好的初始策略，有效的加快了收敛速度。其中算法 A4 中提出的 K-Means 聚类的是性能提升的最关键点，很大程度地利用启发信息使得初始分组更合理化。除此之外，经过与本小组中其他同学的比对，本文提出的针对分组初始化的优化算法，在对收敛速度和最优初始策略的优化效果上是最优的。

但是本文算法仍然存在以下问题可以优化：

- 由于数据集有限，本文的算法都是在小数据集，即 100 个 item 上运行的，因此在大数据集上的性能未知。根据聚类的特点可以预测算法在大数据集上获得初始接将会更加耗费时间。
- 针对论文 [1] 初始化 item 分组和分配的过程，还有第二个薄弱点，即使用贪婪算法进行初始的 item 分配，该算法在大部分情况下可以找到一个相对较优的可行分配策略，但是在某些极端情况下获得的策略会与最优策略相距较远，本文并未对此薄弱点进行优化，后续工作中可采用一致均衡搜索等算法进行优化。
- 本文仅对初始策略进行优化，并未对论文 [1] 中使用到的禁忌搜索进行优化，也并未对算法找到的最优解进行提升（可能已经是最优而无法提升），且论文中提出的局部搜索策略考虑已经较为充分，本文尝试优化但是结果并不理想，这两点是今后可以优化的地方，但是也是较为困难的优化点。

参考文献

[1] Jing Xie et al. “A bi-level optimization model for grouping constrained storage location assignment problems”. IEEE Transactions on Cybernetics 48.1 (2018), pp. 385–398.