COM1005: Machines and Intelligence

Assignment 2

Documentation

Experimental Results

Below are the results comparing the old Bagger1 and new Bagger1, which is called Bagger2. Seeds of “12345”, “24680” and “13579” were used in the testing.

Items (Seed "12345")	Total Space	Minimum Possible Bags Required	Bagger1	Bagger2
			Bags required	
10	506	6	8	6
20	954	10	14	11
30	1401	15	21	16
40	1712	18	25	19
50	2326	24	32	26
100	5194	52	68	59

Table 1 Results of seed “12345” and total items 10, 20, 30, 40, 50 and 100.

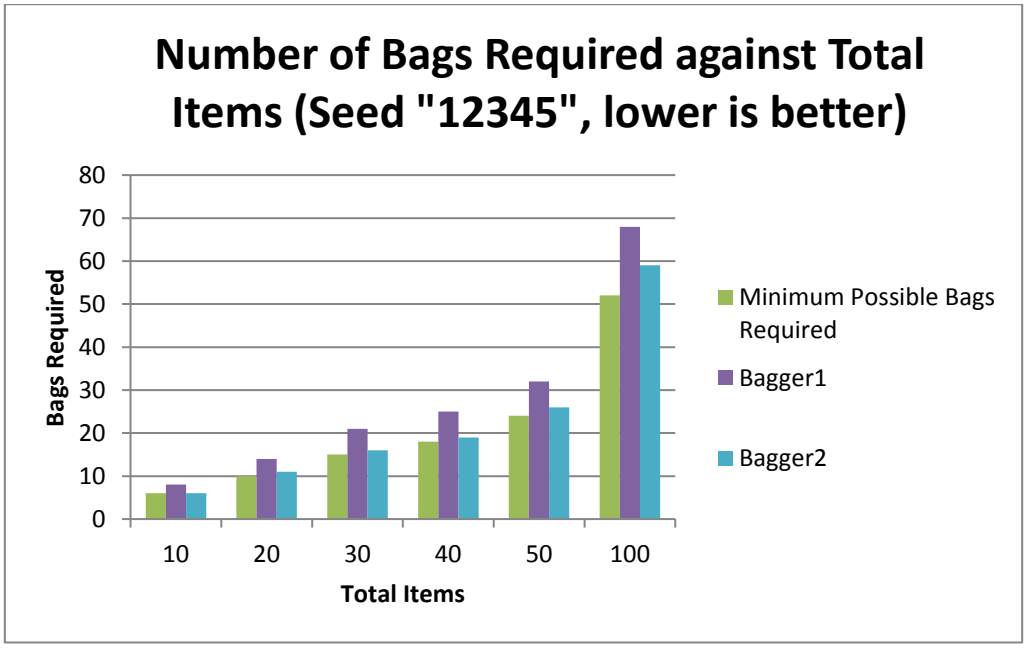

Figure 2 Comparison of the number of bags used in Bagger1 and Bagger2.

Items (Seed "24680")	Total Space	Minimum Possible Bags Required	Bagger1	Bagger2
			Bags required	
10	526	6	10	6
20	866	9	14	10
30	1358	14	20	15
40	1716	18	25	18
50	2238	23	33	24
100	4750	48	65	51

Table 2 Results of seed "24680" and total items 10, 20, 30, 40, 50 and 100.

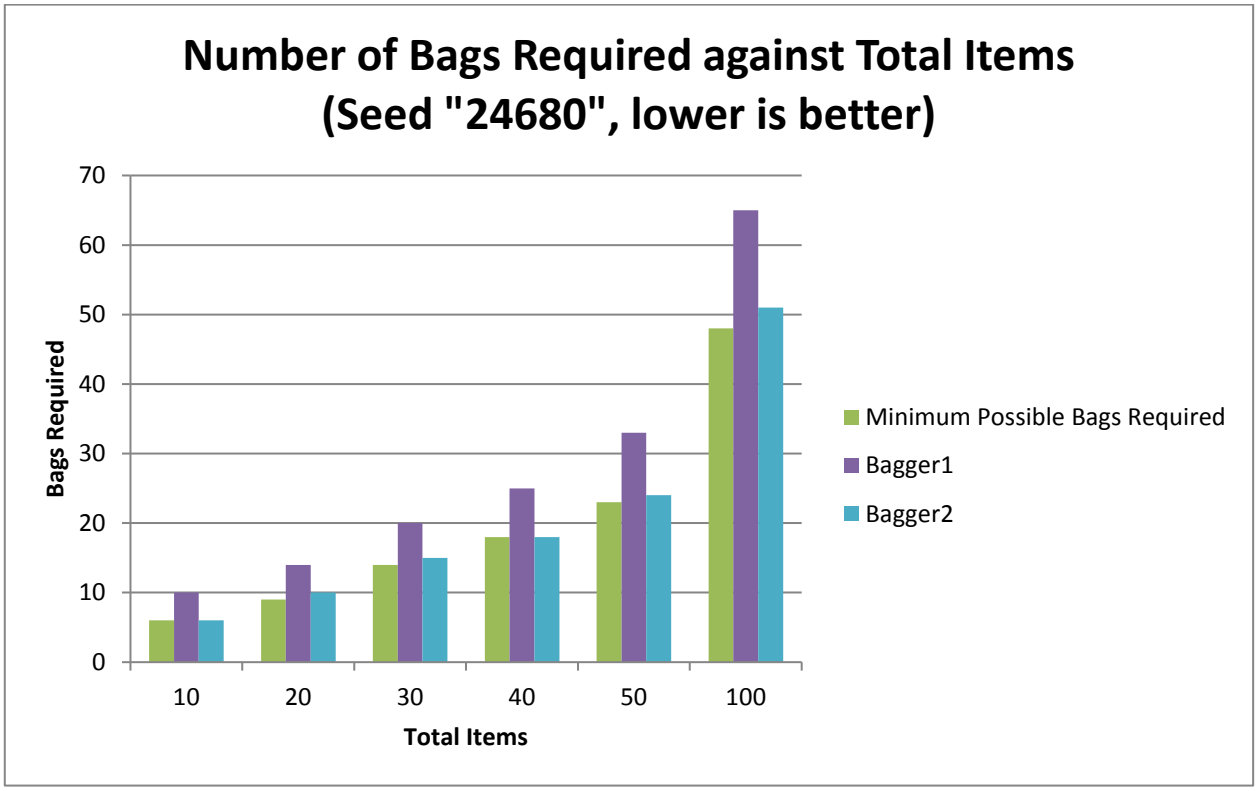

Figure 2 Comparison of the number of bags used in Bagger1 and Bagger2.

Items (Seed "13579")	Total Space	Minimum Possible Bags Required	Bagger1	Bagger2
			Bags required	
10	457	5	8	6
20	959	10	16	11
30	1330	14	21	16
40	1973	20	30	23
50	2519	26	37	29
100	5131	52	74	56

Table 3 Results of seed "13579" and total items 10, 20, 30, 40, 50 and 100.

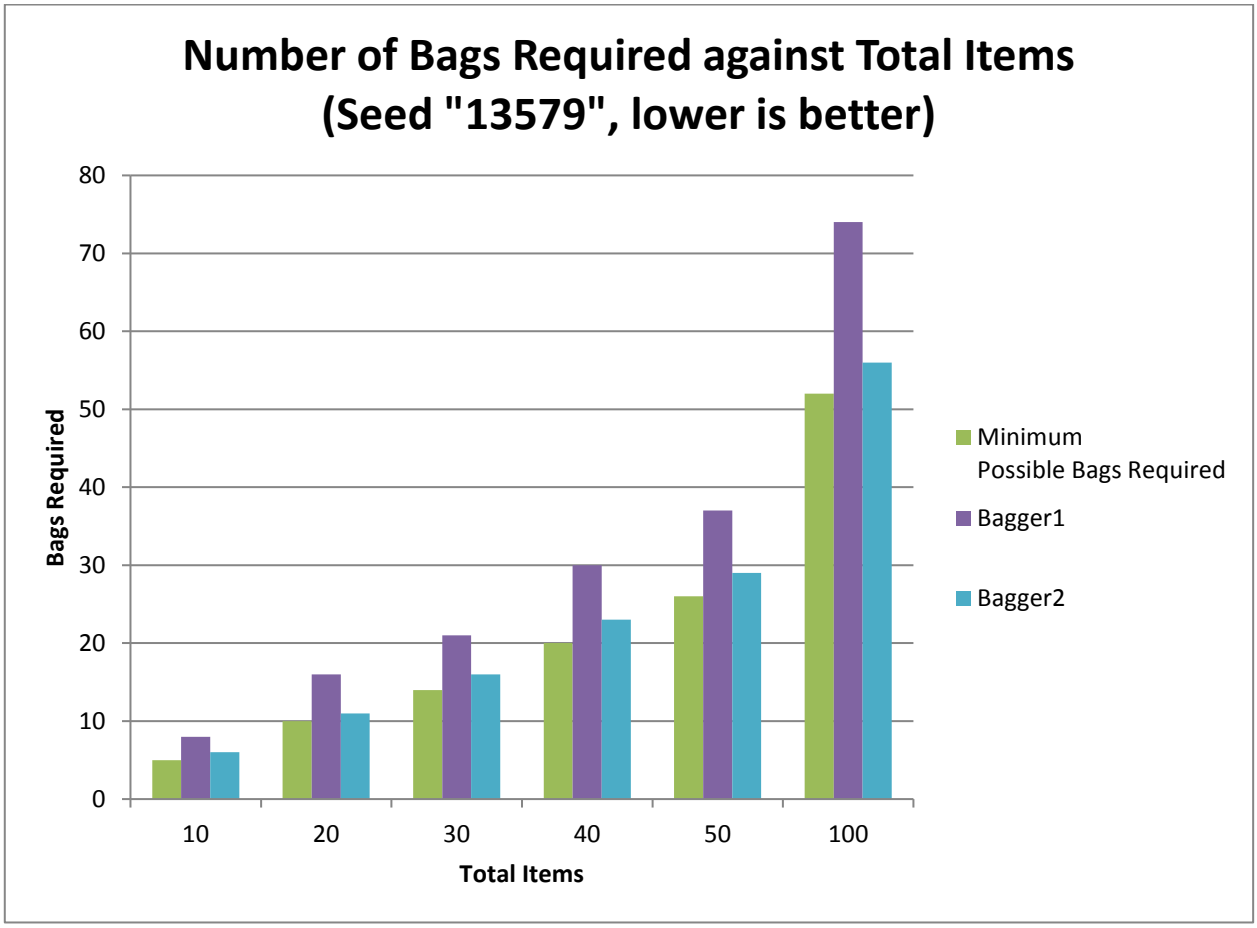

Figure 3 Comparison of the number of bags used in Bagger1 and Bagger2.

Total Items	Total Space	Minimum Possible Bags Required	Bagger1	Bagger2
			Total Bags Required	
750	35916	370	521	402
Items per Bag		**2.03**	**1.44**	**1.87**

Table 4 Overall results from all 3 tests.

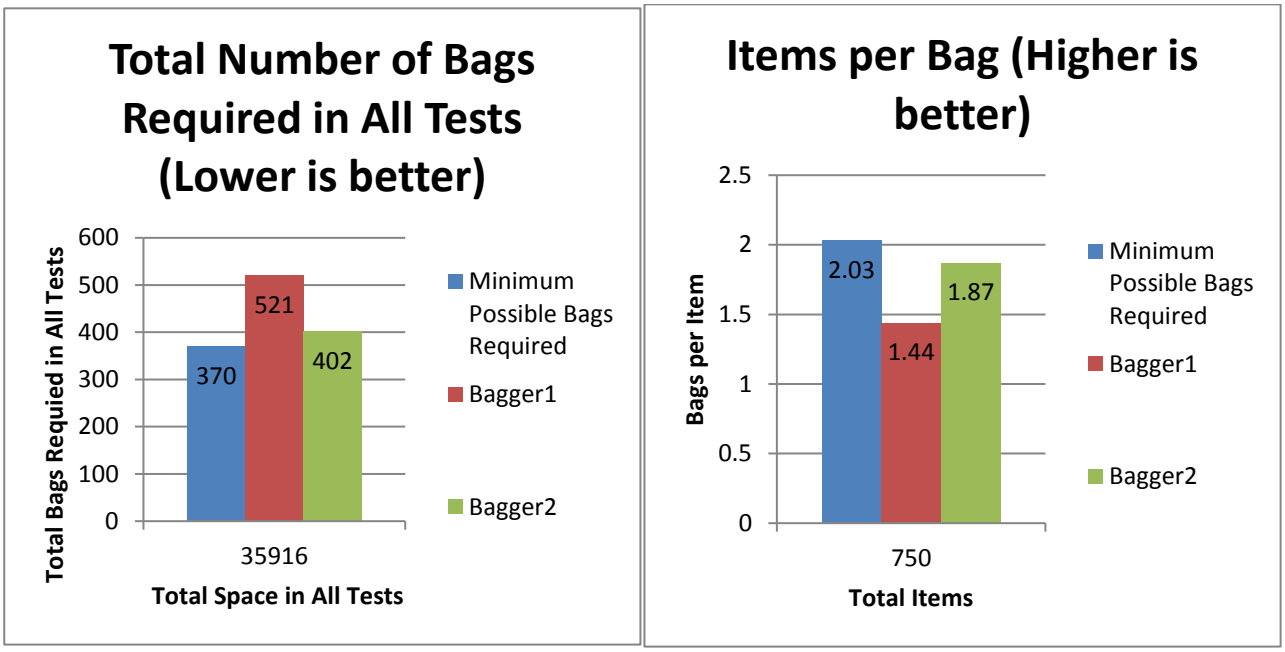

Figure 4 Overall comparison of the number of bags used in Bagger1 and Bagger2.

Efficiency gain of Bagger2 compared to Bagger1

Seed	Total Bags Required		Overall Efficiency Gain (%) $\frac{Bagger1 - Bagger2}{Bagger1} \times 100\%$
	Bagger1	Bagger2	
12345	168	137	18.45
24689	167	124	25.75
13579	186	141	24.19
Average			**22.8**

Table 5. Overall efficiency gain of Bagger2 compared to Bagger2.

According to the three tests above and the overall comparison shown in **Table 4, Table 5** and **Figure 4**, it has been shown that Bagger2 uses fewer bags than Bagger1. The minimum possible bags required are calculated using the formula below:

$$\frac{Total\ space}{100}$$

For example:

$$\frac{5194}{100} = 51.94$$

$$\therefore Minimum\ possible\ bags\ required = 52$$

This is by assuming that every item cost 1 space. The final answer has to be round up because it is only possible to have an integer amount of bags. In practical it is unlikely to have all items of 1 space, therefore, the minimum possible bags required is the best case.

The overall efficiency gain is calculated using the formula shown in **Table 5**, which indicates the difference in number of bags used when comparing Bagger2 and Bagger1. The final result has shown that the Bagger2 is **22.8%** more efficient than Bagger1. Thus, it has been proved that Bagger2 is more efficient than Bagger1.

How Bagger2 works?

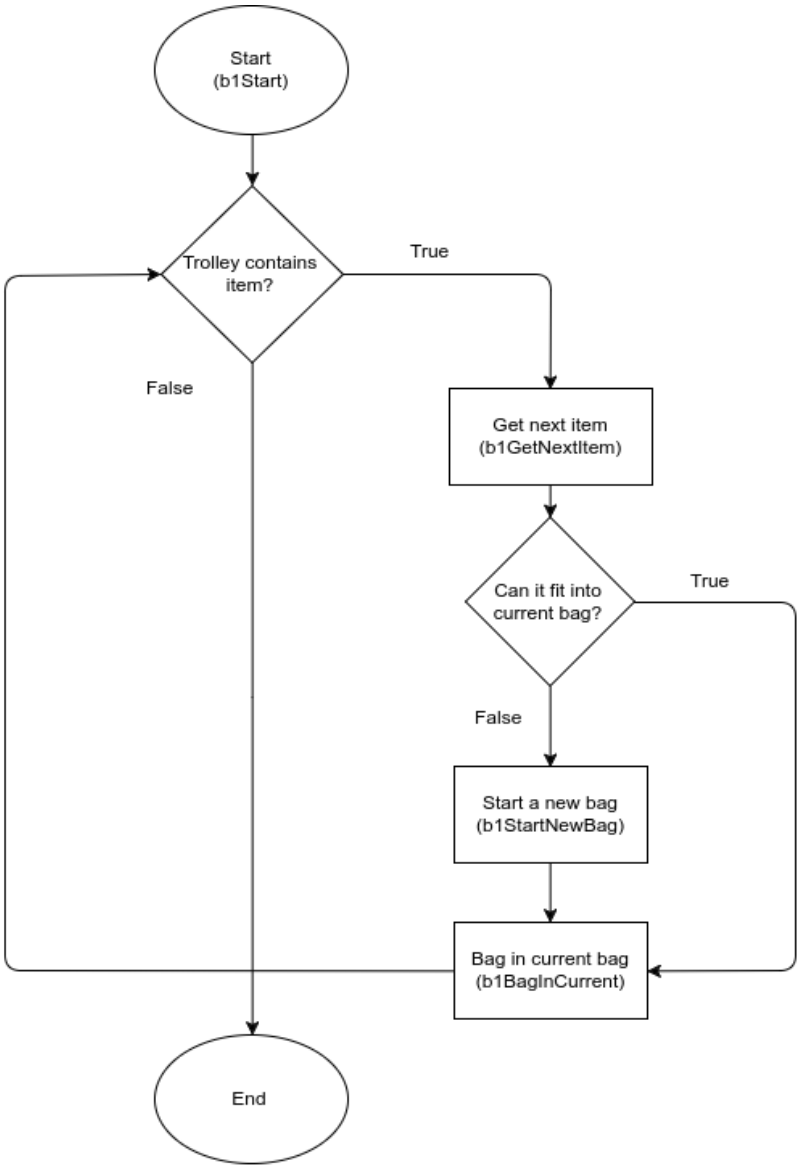

Figure 5 Flow chart of Bagger1

The production system of Bagger1 is simple and it does not optimise the use of available spaces. Below is an output of the initial STM and final STM of seed "12345", items 10 in Bagger1:

Initial STM = [step is start bagging, trolley contains item1 space 51, trolley contains item2 space 80, trolley contains item3 space 41, trolley contains item4 space 28, trolley contains item5 space 55, trolley contains item6 space 84, trolley contains item7 space 75, trolley contains item8 space 2, trolley contains item9 space 1, trolley contains item10 space 89]

Final STM = [bag 1 contains item1, bag 2 contains item2, bag 3 contains item3, bag 3 contains item4, bag 4 contains item5, bag 5 contains item6, bag 6 contains item7, bag 6 contains item8, bag 6 contains item9, step is get next item, bag 7 contains item10, current bag no 7 space 11]

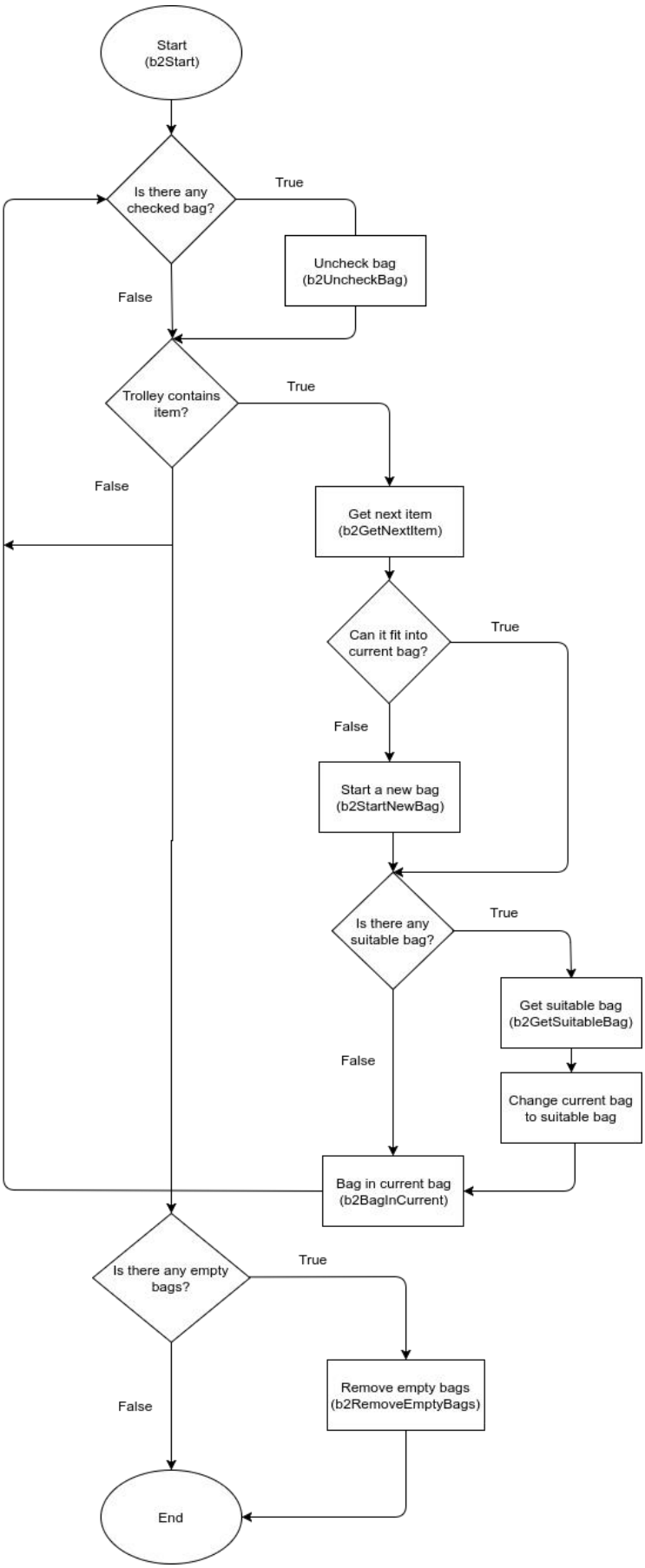

Figure 6 Flow chart of Bagger2, the improved version of Bagger1.

Figure 5 and **Figure 6** have shown the flow chart of Bagger2 and Bagger1 respectively. The main difference is that the Bagger2 has more checking processes, which ensure that the next item is placed into the most suitable bag. To improve the efficiency of the bagger system, it must make the most optimal use of the remaining spaces in previous bags.

The Bagger2 production system has three new classes, which are **b2GetSuitableBag**, **b2RemoveEmptyBag** and **b2UncheckBag**. Below are the functions and how each class works.

- b2GetSuitableBag : Loop through all bags with remaining spaces left, this includes the current bag. Then, it will compare the current bag against the previous bags. If one of the previous bags has enough space for the next item and its remaining space after bagger the next item is closest to 100, then the current bag is changed to that previous bag. Every previous bag that has been checked will be saved as **bag ?N space ?S (checked)** in the STM.

- b2UncheckBag : After bagging the item into the current bag, the system will uncheck all the previous bags by changing **bag ?N space ?S (checked) back** to **bag ?N space ?S** so that they can be checked again in the next loop.

- b2RemoveEmptyBag : The loop will continue until there are no more items in the trolley. When the loop terminates, if there are any bags with 0 space left in the STM, which are saved as **bag ?N space 0,** then it will be removed from the STM.

The classes above are implemented to improve the Bagger1 system, which are proved to be useful and greatly increased the efficiency and reduced the number of bags used.

Below is the output of initial STM and final STM of seed "12345", items 10 in Bagger2:

Initial STM = [step is start bagging, trolley contains item1 space 51, trolley contains item2 space 80, trolley contains item3 space 41, trolley contains item4 space 28, trolley contains item5 space 55, trolley contains item6 space 84, trolley contains item7 space 75, trolley contains item8 space 2, trolley contains item9 space 1, trolley contains item10 space 89]

Final STM = [bag 1 contains item1, bag 2 contains item2, bag 1 contains item3, bag 3 contains item4, bag 3 contains item5, bag 4 contains item6, bag 5 contains item7, bag 1 contains item8, bag 1 contains item9, total 6 bags, step is uncheck bag, bag 6 contains item10, current bag 6 space 11, step is get next item, bag 2 space 20, bag 5 space 25, bag 3 space 17, bag 4 space 16, bag 1 space 5]

Below is a runtime comparison of the Bagger1 and Bagger2:

Bagger1:

STM= [trolley contains item3 space 80, bag 1 contains item1, step is get next item, bag 2 contains item2, current bag no 2 space 0]

Deletions:
step is get next item
trolley contains item3 space 80
Firing GET-NEXT-ITEM
in context {?S=80, ?I=item3}
GET-NEXT-ITEM remarks:
bagging item3

STM= [bag 1 contains item1, bag 2 contains item2, current bag no 2 space 0, step is bag item, item to bag item3 space 80]

Deletions:
current bag no 2 space 0
Firing START-NEW-BAG
in context {?BS=0, ?N=2, ?NB=3}
START-NEW-BAG remarks:
Starting bag 3

STM= [bag 1 contains item1, bag 2 contains item2, step is bag item, item to bag item3 space 80, current bag no 3 space 100]

Deletions:
step is bag item
item to bag item3 space 80
current bag no 3 space 100
Firing BAG-IN-CURRENT
in context {?BS=100, ?RS=20, ?I=item3, ?S=80, ?N=3}
BAG-IN-CURRENT remarks:
item3 in bag no 3

The old Bagger1 does not check previous bags for space. If the next item cannot fit into the current bag, then it will start a new bag.

Bagger2:

STM = [trolley contains item3 space 80, bag 1 contains item1, total 2 bags, step is uncheck bag, bag 2 contains item2, current bag 2 space 0, step is get next item, bag 1 space 80]

Deletions:
    step is get next item
    trolley contains item3 space 80
Firing GET-NEXT-ITEM
    in context {?S=80, ?I=item3}
GET-NEXT-ITEM remarks:
    Bagging item3

STM = [bag 1 contains item1, total 2 bags, step is uncheck bag, bag 2 contains item2, **current bag 2 space 0**, **bag 1 space 80**, step is get suitable bag, step is bag item, item to bag item3 space 80]

Deletions:
    step is uncheck bag
    bag 1 space 80
    current bag 2 space 0
Firing GET-SUITABLE-BAG
    in context {?BS=0, ?PB=1, ?NB=2, ?S=80, ?SBS=80, ?I=item3, ?NBS=0, ?SB=1, ?PBS=80, ?N=2}
GET-SUITABLE-BAG remarks:
    Bagging item3 in bag 1

STM = [bag 1 contains item1, total 2 bags, bag 2 contains item2, step is get suitable bag, step is bag item, item to bag item3 space 80, **bag 1 space 80 (checked)**, **current bag 1 space 80**, **bag 2 space 0 (checked)**]

Deletions:
    step is get suitable bag
    step is bag item
    item to bag item3 space 80
    current bag 1 space 80
    bag 1 space 80 (checked)
Firing BAG-IN-CURRENT
    in context {?BS=80, ?RS=0, ?I=item3, ?S=80, ?N=1}
BAG-IN-CURRENT remarks:
    item3 in bag 1

The Bagger2 check previous bags for space every time after getting the next item. Notice the **bold word** in the code above, initially the current bag is **bag 2 space 0**, which has no remaining space left for next item. In the STM there is a **bag 1 space 80**, which has enough space for the next item. Therefore, the current bag is changed to **bag 1** and bags item3 into it.

Discussion

According to the results shown in **Figure 1**, **Figure 2**, **Figure 3** and **Figure 4**, it has been proved that the Bagger2 has better performance and efficiency when compared to the Bagger1. The data in **Table 4** shows that the Bagger2 uses an average of **1.87 items per bag** and the Bagger1 uses **1.44 items per bag** in the three tests above. The best possible result would be **2.03 items per bag**. Since pre-sorting items in the trolley are restricted, this Bagger2 result might be the result obtainable. If it would be possible to pre-sort the items, the result might use fewer bags because it could find the right combinations of items that add up to 100 spaces and put them into one bag. The problem of Bagger2 is that even by checking the previous bags and putting the item in the most suitable bag, it might not be the most optimal way to put, for example, **item1** and **item7** together into one bag. This might also be the reason of why there are a lot of bags with very little remaining space in the final STM.

Without pre-sorting the items:

- Item1 space 20
- Item2 space 100
- Item3 space 70
- Item4 space 30

Results:

- Bag1
  - Item1 space 20
  - Item3 space 70
  - Space left: **10**
- Bag2
  - Item2 space 100
  - Space left: 0
- Bag3
  - Item4 space 30
  - Space left: **70**

Pre-sort the items:

- Item2 space 100
- Item3 space 70
- Item4 space 30
- Item1 space 20

Results:

- Bag1
  - Item2 space 100
  - Space left: 0
- Bag2
  - Item3 space 70
  - Item4 space 30
  - Space left: 0
- Bag3
  - Item1 space 20
  - Space left: **80**

The example above has shown the advantages of pre-sorting the items. By pre-sorting the items, the amount of remaining space wasted could be minimized. It is better to have a total of **80** remaining space in one bag rather than splitting it into **10** and **70**. If the next item **Item5** occupies **80 spaces**, then the Bagger2 will start a new bag because there are no previous bags with available space. However, if the items are pre-sorted, then **Bag3** would have enough space for **Item5**, and three of the bags would be full with no space wasted. If the number of items is huge, then the impact of pre-sorting could be significance. This is because the item space is fixed, and the way they are organized may affect the outcome, which is similar to the **Two Monks Paradox (Image 1)**.

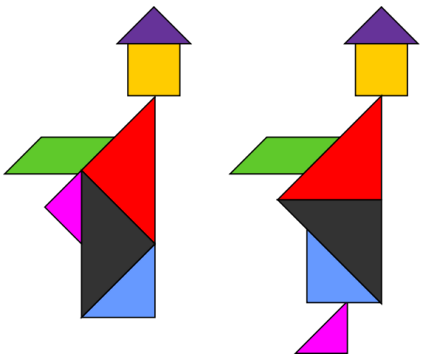

Image 1. A tangram paradox of two monks. (Source: Wikipedia)

If each of the shapes above represents an item, then it is proved that different combinations of items in a bag would produce a different result. If this hypothesis is true, then there must be an optimal way to arrange the items so that they will use the least possible bags. However, in practical, if the number of items is huge, it might be too complicated and it could take a long time and for Robbie the robot to search for the optimal way of bagging the items.

Conclusion

Overall, according to the three test results shown in **Figure 1**, **Figure 2**, **Figure 3**, and the overall result in shown **Table 4** and **Figure 4**, it has been proved that the Bagger2 has better efficiency than Bagger1. The Bagger2 uses an average of **1.87 items per bag** while the Bagger1 uses an average of **1.44 items per bag**, and the minimum possible is **2.03 items per bag**. The results has also proved that by selecting the most suitable bag from previous bags and bagging the item could greatly reduce the amount of bags used. If pre-sort were possible, the result could be further improved as it will reduce the amount of remaining space wasted.