Here are running time of some of test cases:
S-k-20-30: 1s
S-k-20-35: 1s
S-k-30-50: 1s
S-k-40-60: 1s
S-k-35-65: 3s
S-k-40-80: 21s
S-k-50-95: unknown

s-X-12-6: 1s
S-rg-8-10: 1s
S-rg-31-15: 1s
S-rg-40-20: 1s
S-rg-63-25: 1s
S-rg-118-30: 52s
S-rg-109-35: unknown

Here are how I implement my program:

First of all, I traverse all the elements in all the subsets to see if there are unique element that appear only once in all the subsets. This means that subsets that contains the unique element is always needed in my answer. However I notice that, this does help much because almost all of elements have duplicate.

Also, before I move into the backtracking stage, I sort the all the subsets based the number of elements in each subset. So the biggest subset is at the front and the smallest subset at the end. The advantage for this optimization is that those bigger subset contains more elements and thus have a higher probability to be in the result and help the program find the result earlier.

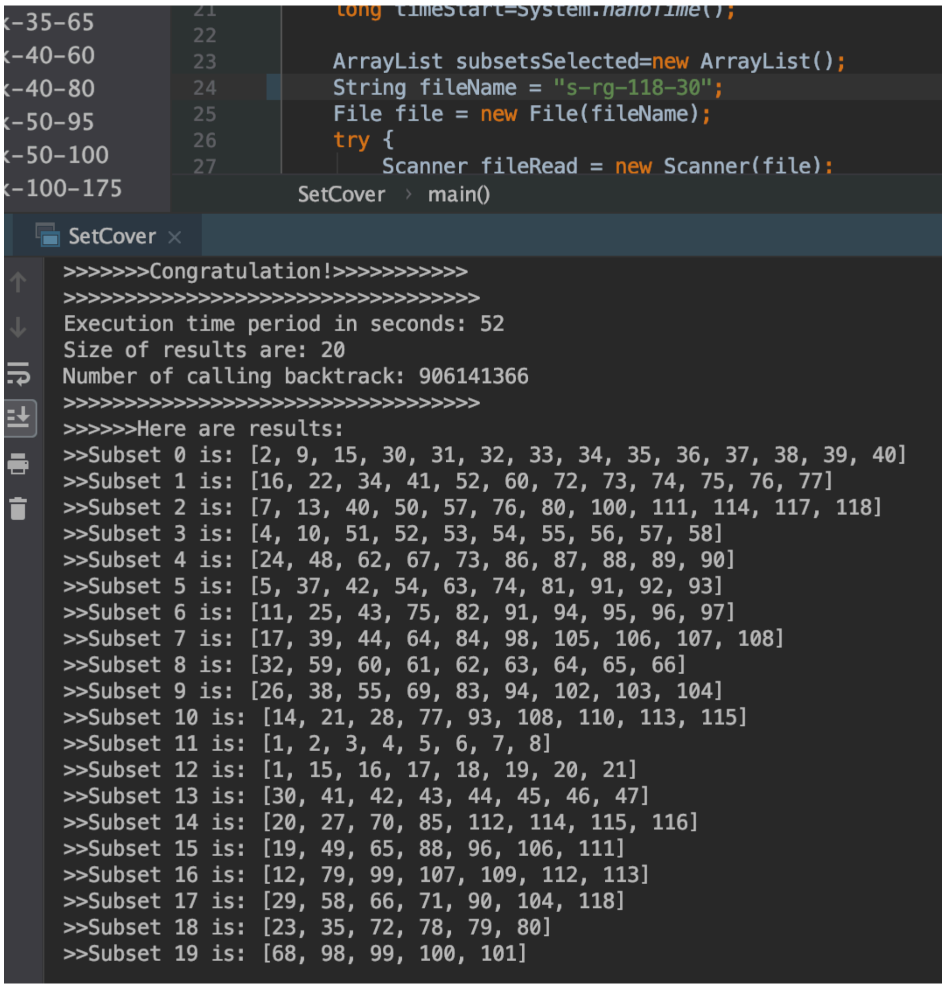
A big achievement I did is making the complexity of isValid() method O(1). By having an array whose length is the length +1. The first element of the array is the number of non-Zero entries in the array. Then each element of array is the number of each element of universe in the partial solution. I revise this array every time I add a subset into partial solution and remove. So in isValid method, it just check whether the first element of the array is 0 or not.

The array representing the number of every element in current partial solution mentioned above has another purpose in constructCandidates() method. In this method, I have to choose candidates among the rest of subsets that is not in the partial solution. By checking whether every element of that possible subset is already coved in the partial solution, I can say whether a subset is not needed and I will not put that into candidates. By also I can prune a lot of unnecessary branches.

For test cases whose name start with “s-k”, from my observation, many of subsets have only one elements, and that elements already appear in another subset. That mean, those subsets could be discarded with going into the backtracking. This makes the problem much less complex. For instance, in the test “s-k-40-80”, and half of all the subsets have only one elements and already covered in other subsets. So I can delete those subsets. Then there are only 40 subsets left to move into the backtracking stage.

When the partial solution at a certain stage have 6 subsets and the temporary result I saved also has size of 6, I intentionally avoid it to backtrack to make the partial solution of size 7. No matter whether we can get a result from that partial solution, the length of that is always bigger or equal to the result. It doesn’t help us with the minimum size of set cover. By pruning many unnecessary branches, the time the program took significantly dropped.

And here are screenshots of console when I run the big test:
s-rg-118-30:



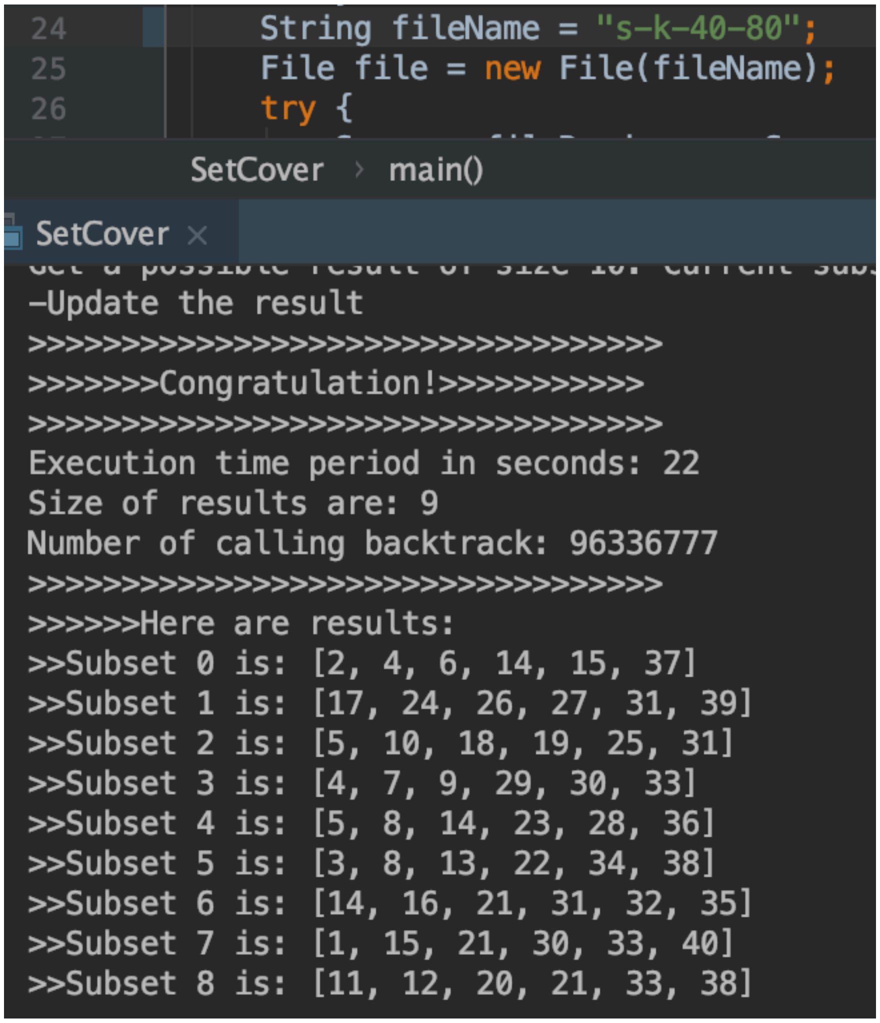
```
-35-65
-40-60
-40-80
-50-95
-50-100
-100-175
```

```
        long timeStart=System.nanoTime();

        ArrayList subsetsSelected=new ArrayList();
        String fileName = "s-rg-118-30";
        File file = new File(fileName);
        try {
            Scanner fileRead = new Scanner(file);
```

```
>>>>>>>Congratulation!>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Execution time period in seconds: 52
Size of results are: 20
Number of calling backtrack: 906141366
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>Here are results:
>>Subset 0 is: [2, 9, 15, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]
>>Subset 1 is: [16, 22, 34, 41, 52, 60, 72, 73, 74, 75, 76, 77]
>>Subset 2 is: [7, 13, 40, 50, 57, 76, 80, 100, 111, 114, 117, 118]
>>Subset 3 is: [4, 10, 51, 52, 53, 54, 55, 56, 57, 58]
>>Subset 4 is: [24, 48, 62, 67, 73, 86, 87, 88, 89, 90]
>>Subset 5 is: [5, 37, 42, 54, 63, 74, 81, 91, 92, 93]
>>Subset 6 is: [11, 25, 43, 75, 82, 91, 94, 95, 96, 97]
>>Subset 7 is: [17, 39, 44, 64, 84, 98, 105, 106, 107, 108]
>>Subset 8 is: [32, 59, 60, 61, 62, 63, 64, 65, 66]
>>Subset 9 is: [26, 38, 55, 69, 83, 94, 102, 103, 104]
>>Subset 10 is: [14, 21, 28, 77, 93, 108, 110, 113, 115]
>>Subset 11 is: [1, 2, 3, 4, 5, 6, 7, 8]
>>Subset 12 is: [1, 15, 16, 17, 18, 19, 20, 21]
>>Subset 13 is: [30, 41, 42, 43, 44, 45, 46, 47]
>>Subset 14 is: [20, 27, 70, 85, 112, 114, 115, 116]
>>Subset 15 is: [19, 49, 65, 88, 96, 106, 111]
>>Subset 16 is: [12, 79, 99, 107, 109, 112, 113]
>>Subset 17 is: [29, 58, 66, 71, 90, 104, 118]
>>Subset 18 is: [23, 35, 72, 78, 79, 80]
>>Subset 19 is: [68, 98, 99, 100, 101]
```

s-k-40-80:



```
        String fileName = "s-k-40-80";
        File file = new File(fileName);
        try {
```

```
Get a possible result of size 10. Current sub
-Update the result
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>Congratulation!>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Execution time period in seconds: 22
Size of results are: 9
Number of calling backtrack: 96336777
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>Here are results:
>>Subset 0 is: [2, 4, 6, 14, 15, 37]
>>Subset 1 is: [17, 24, 26, 27, 31, 39]
>>Subset 2 is: [5, 10, 18, 19, 25, 31]
>>Subset 3 is: [4, 7, 9, 29, 30, 33]
>>Subset 4 is: [5, 8, 14, 23, 28, 36]
>>Subset 5 is: [3, 8, 13, 22, 34, 38]
>>Subset 6 is: [14, 16, 21, 31, 32, 35]
>>Subset 7 is: [1, 15, 21, 30, 33, 40]
>>Subset 8 is: [11, 12, 20, 21, 33, 38]
```