John Beighle
CSCE 686
HW5

1). (15 points) Evaluate AFIT SCP Solver

Subtask 1a: Consider SCP test suites: OR-Library, papers. Select at least three test problems - small, medium, large dimension (show details). Generate and discuss results.

Evaluating both the C++ and Java versions of the AFIT SCP Solver. The following result were obtained.

	Graphs		
Type	*small*	*medium*	*large*
Nodes	6	6	6
Sets	5	64	25
Saturation	dense	sparse	dense
C++ Time	800	7	0
Java Time	0.1	17	55

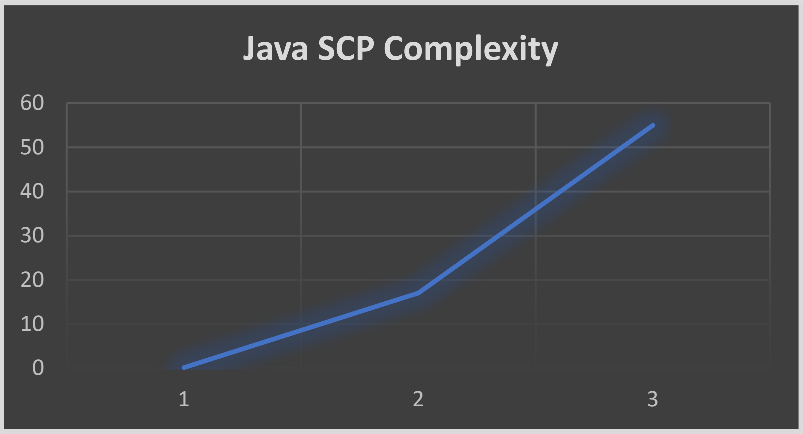


Figure 1

Cormode et al [3] reported very similar result to result obtained in this pedagogical exercise. Their work focused more on optimizing large datasets and made advancements towards dealing with IO. Their algorithm “rewrites a set Si to disk or memory in a “reduced” form, with the covered elements removed, if it is not immediately added to the solution. Applying this improvement would potentially be useful in other cases, such as the greedy algorithm (especially the multiple pass version), although this might not make enough of a difference to the general poor performance.”[3] They report that the Christofides algorithm is about as good as it gets in term of overall performance. For smaller datasets, with our experiments, it seems to be analogous at least. The data above is at least pointing in the direction of 2^n bound.

John Beighle
CSCE 686
HW5
Problems experienced with the AFIT sourcecode:

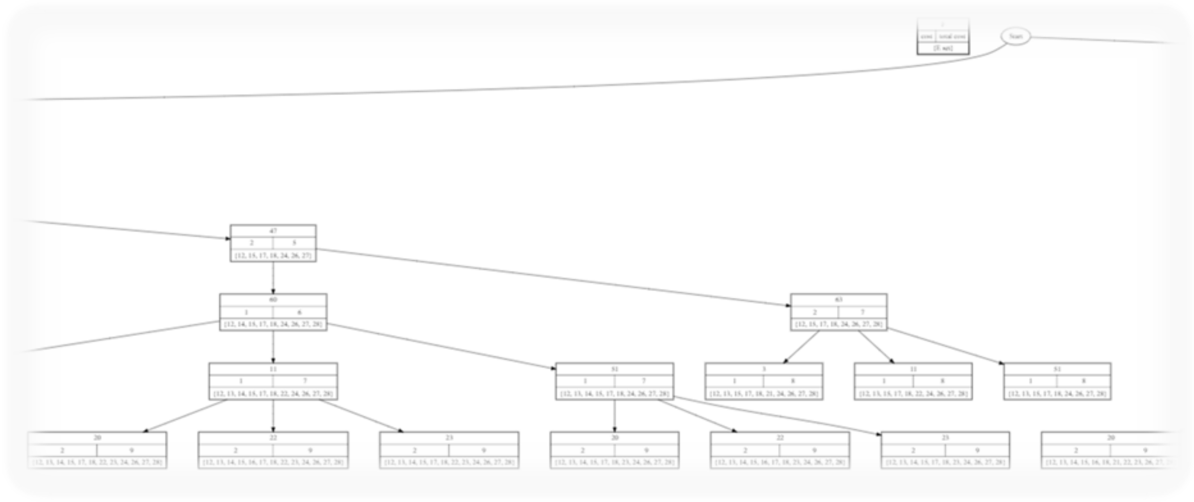
1. Cpp doesn’t work unless there’s a blank line at the end of a file
2. Java doesn’t work unless it’s recompiled – doesn’t calculate durations
3. Java duration calculations are wrong because it includes user input times
4. Java duration calculations are wrong because it writes output files in sub methods
5. Java solutions are sometimes wrong. Example
6. C++ example would not process large graphs. It hung on multiple graphs.

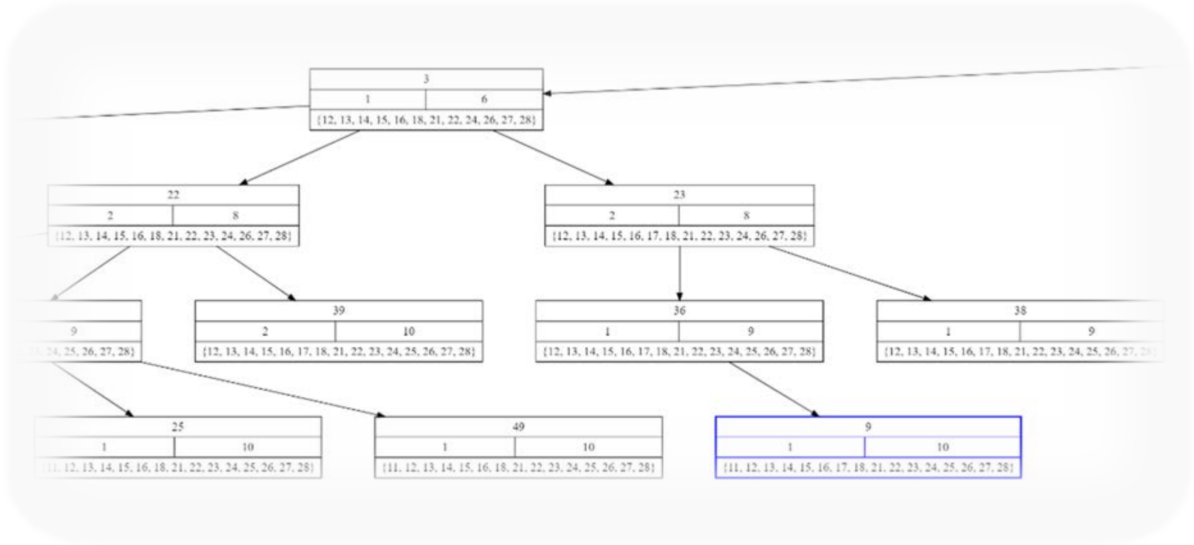
Input file contents of small:
{ 1 2 3 4 5 6 } {   1: ({ 1 2 3 }, 1)   2: ({ 2 3 4 }, 2)   3: ({ 3 4 5 }, 3)   4: ({ 1 3 5 }, 4)   5: ({ 3 5 6 }, 5) }
The Solution Set: [1, 5]

John Beighle
CSCE 686
HW5

Subtask 1b: Generate an explicit graph search tree diagram for medium SCP test example (limit search node presentation)

Images below are small cut segments of Med graph's AFITs Java DOT in GraphViz. Graph was very wide.





Subtask 1c: Discuss Complexity of SCP PD and Christofides' algorithm, and complexity of AFIT SCP Solver implementation.

AFIT SCP implements the preliminary reduction tests of section 4.2 Christofides [1]. Both provided implementations of Christofides employ the tableau sorting blocks based on sets that cover an element and sorting within blocks based on set costs. For these aprior sorts a minimum such value should be achieved with an array-based heap implementation of a priority queue. The AFIT Java solver merely utilizes basic Java collections and will suffer some performance penalties as the datasets grow. These pre-sorts complexities will be worse than what we find in standard C++11 qsort for a priority queue implementation. AFIT's java sorts uses standard collection mechanisms like ArrayList. C++11 standard requires that the complexity of sort to be O(Nlog(N)) in the worst case. Previous versions of C++ such as C++03 allow possible worst-case scenarios of O(N^2). Only average complexity was required to be O(N log N). Our application design document stated a problem domain complexity of O(mnlogn) and space O(m2n) with the upper bound on data size at 2^n for worst case exhaustive tree search.

The AFIT implementations with the tableau reductions should provide optimal solutions in polynomial time if there is enough time to find them. These times, as has been stated above, align to some degree to published baseline complexities. Christofides of section 4.4 has a table of computing times for random trees. These are comparable to what is discovered here.

The decision version of set covering is NP-complete, and the optimization/search version of set cover is NP-hard.[2]

2). (10 points)

Subtask 2a: Define a new SCP Heuristic (English and math/symbolic)

This is not new but a very famous one and one that is widely used still: The Chvatal greedy algorithm builds a cover by repeatedly choosing a set s that minimize the weight ws divided by number of elements in s not yet covered by chosen sets. It stops and returns the chosen sets when they form a cover:

```
greedy-set-cover($\mathcal{S}$, $w$)
1. Initialize $C \leftarrow \emptyset$. Define $f(C) \doteq |\cup_{s \in C} s|$.
2. Repeat until $f(C) = f(\mathcal{S})$:
3.     Choose $s \in \mathcal{S}$ minimizing the price per element $w_s/[f(C \cup \{s\}) - f(C)]$.
4.     Let $C \leftarrow C \cup \{s\}$.
5. Return $C$.
```

The problem with this heuristic is that the cost decision and log factor can make errors when choosing optimal paths by the fact that where I = {} the Cost(S_i) / |S – I| can introduce cost errors choosing minimal cost.[9]

Subtask 2b: Define a SCP approximation algorithm in Talbi's form. Discuss approximation results with optimal results (theory?)

Naive heuristic [4]. Proceeds as follows:

Sort the (indices i of the) sets Si into descending order according to |Si|. For each Si in this order, until C = X: – If |Si \ C| > 0: add i to Σ and update C. In the worst case this heuristic cannot provide an approximation with approximation ratio better than n/6. Consider an instance in which Si = {2i − 1, 2i, . . . , 2k + i} for some k and all i ≤ k. Now, |Si| = 2k + 2 − i, so the naive algorithm will process the sets in the order S1, S2, . . . , Sk, each time adding the set Si to the solution, because it contains one uncovered element, 2i + 1. The optimal solution, however, comprises just S1 and Sk. Therefore the ratio of the sizes of the naive and optimal solutions is k/2 = n/6.

Subtask 2c: If possible, compare results with two different SCP algorithm implementations.

Comparisons are within reasonable range of both Christofides random graph results [8 pg 46] and Cormode et al. [3]. However, the Christofides results were from slower hardware and accounts for overall slower order of magnitude.

John Beighle
CSCE 686
HW5

3). (5 points)

Subtask 3a: Have “good" software engineering principles (design and implementation) been employed in the AFIT SCP Solver? Regarding the implementation, discuss: Ease of understanding of code (suggested modifications?)
Ease of use of standard interface (modifications?)

This section will focus on a response in terms of the Java program. The most important software engineering principle, ”measure twice and cut once” was not followed sufficiently because the Java implementation, at least, produced erroneous results on occasion. This carries into another important principle that was violated to ensure usability with adequate testing. Additional comments on common areas:

1. Don’t repeat yourself. Good marks in this area.
2. Occam’s Razor. Good marks here.
3. Keep it simple. Poor marks, little to no comments, a lot of object names were obfustacted
4. Big design up front. Good marks here.
5. Avoid premature optimization. Poor marks. The ADT design lacked optimization.

In terms of ease of understanding the code, again, not the easiest to follow in terms of mapping to design and pseudocode and a few of the object names were unhelpful, e.g. argx, argy. The simplest and best suggestion, since a lot of effort went into this solution, is just to go the extra step and provide comments and explicit mapping to the pseudocode/design.

4). (5 points)

Subtask 4a: Define a real-world (not pedagogical) discrete combinatoric optimization SCP NPC problem associated with an Air Force application (Networks, UAVs, Robotics, Sensors, …) – examples pg 48 Christofides.

Example: The US Air Force needs to buy a certain amount of varied supplies and there are suppliers that offer various deals for different combinations of materials (Supplier A: 2 tons of steel + 500 tiles for $x; Supplier B: 1 ton of steel + 2000 tiles for $y; etc.). You could use set covering to find the best way to get all the materials while minimizing cost.

References:
[1] Chrisfides Chp3 SCP
[2] Korte, Bernhard; Vygen, Jens (2012), Combinatorial Optimization: Theory and Algorithms (5 ed.), Springer, ISBN 978-3-642-24487-2
[3] Set Cover Algorithms For Very Large Datasets Graham Cormode, Howard Karloff, Anthony Wirth
[4] F. Chierichetti, R. Kumar, and A. Tomkins. Max-Cover in Map-Reduce. In Proceedings of the 19th International Conference on World Wide Web, pages 231–40. ACM, 2010.
[5] U. Feige. A threshold of ln n for approximating set cover. Journal of the ACM, 45(4):634–52, 1998.
[6] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. Profiling high frequency accident locations using association rules. In Proceedings of the 82nd Annual Transportation Research Board, page 18pp,
[7] Talbi: Section1.7.2.1/.2/.3
[8] Christofides Ch3 SCP
[9] F. Chierichetti, R. Kumar, and A. Tomkins. Max-Cover in Map-Reduce. In Proceedings of the 19th International Conference on World Wide Web, pages 231–40. ACM, 2010.
[10] Greedy Set-Cover Algorithms (1974-1979, Chv´atal, Johnson, Lov´asz, Stein)