EEL 6935: ROBITICS AND AI MINI-PROJECT 1

Nam Nguyen
Department of Electrical Engineering
Department of Statistics and Mathematics
University of South Florida
Tampa, USA
{namnguyen2,chang5}@usf.edu

ABSTRACT

1 GRAPHICAL ALGORITHM

Problem: Serena wants to host a party and to determine whom to be invited. She has n persons to choose from, and she has made up a list for the pairs of these n persons knowing each other (i.e. not necessarily all know each other). She wants to invite as many as possible persons, subject to the constraints: for each person attending the party (i) at least $k << n$ other persons who know (ii) at least k persons in the party who do not know.
(a) Please develop the mathematical model to solve the problem.
(b) Please develop the algorithm which efficiently solve the problem.
Mathematical Assessment: Now we need to clearly define two scenarios, which are:
(i) if a person A knows person B, then person B also know person A. In this case, the bidirectional edges are linked two people in the know, forming an undirected graph G. Hence, we simply mathematically restate the problem as follows:
Given a graph $G(V, E)$ of n vertices. Find the largest sub-graph G', such that: for every node v in G', $\deg(v) \geq k$ and $\deg(v) \leq (n-k)$.

(ii) if a person A knows person B, then person B *not necessarily* know person A. (This situation happens frequently in fact). Thus, graph G is a directed graph. We denote the relation ”A knows B” as $A \rightarrow B$, resulting in out-degree $\deg^{+}(A)$. While a person knowing A ascents one unit in $\deg^{-}(A)$. Mathematically, we have:
Given a graph $G(V, E)$ of n vertices. Find the largest sub-graph G', such that: for every node v in G', $\deg^{+}(v) \geq k$ and $\deg(v)^{+} \leq (n-k)$.

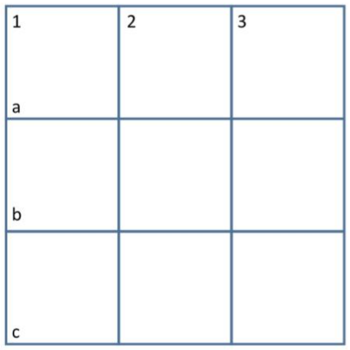
Algorithm 1 Party Planner

```
initialize $G(V, E)$
while $v$ in $V$:
$V^{-} = \emptyset$                                  (Create a blank array to store negative nodes)
    if $\deg(v) < k$ or $\deg(v) > (n-k)$            (Finding negative nodes)
        $V^{-} \leftarrow \{v\}$
            if $V^{-} \neq \emptyset$ random select $v^{-} \in V^{-}$ (Only one node is removed each iteration)
            $V \leftarrow V \setminus \{v^{-}\}$                 (Pruning)
            else return $V$              (Return result if there is no negative node left)
            break
```

Algorithmic Assessment: We proposed a greedy algorithm for the given problem, where construct a solution piece by piece by always considering the next ”best” step. The ”best” here means offering the most feasible and beneficial for the objective. Besides, we find that it is more convenient to tackle the problem by a (backward) pruning process, in stead of (forward) reconstructing a new graph. Thus, a vertex v in the initial graph G must be eliminated if: (i) $\deg(v) < k$ or $\deg(v) > (n-k)$ (ii) $\deg^+(v) < k$ or $\deg^+(v) > (n-k)$, since the degree of such vertex (negative node) will be always too small or too large when we trim the graph. We proposed a algorithmic solution as in Algorithm 1. The time complexity for computing the degree of nodes is $O(n^2)$. Thus, by running iteration over all n nodes yields time complexity of $O(n^2)$.

2 Search Algorithm

Problem: Consider a crossword puzzle as the Figure 1 by using the 24 candidate words on the right-hand side of the figure. Each of the vertical labels, 1, 2,3, and the horizontal labels, a, b, c, must be a word from the candidates. Please develop the algorithm to find all possible solutions, and identify the complexity of your algorithm. Is your algorithm computationally efficient based on number of steps in search (with calculations)? You must specify details such as knowledge representation, pruning methods, etc.



Words to Choose:

add, age, aid, aim, air, are, art, bad, bat, bar, bee, but, cow, dig, ear, eel, get, god, mug, oak, sad, sea, toe, tow

Figure 1: Crossword Puzzle

Problem Assessment: The first assessment about the given problem is that if we consider the puzzle as a 3×3 matrix S, then the transposed matrix S^T is also a solution. Thus, we approach this problem by filling words row-wisely to find S, then transpose the solution to find the corresponding result S^T. Notice that if S is symmetric matrix, then $S = S^T$. Besides, we consider two scenarios, where *with* and *without* repetition of words. We only find the solution once the repetition allowed, whereas there is no solution for the latter case.

Algorithm: We tackle the problem by brute-force search. First, we initialize an array for storing solution, says S:

$$S = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}.$$

To solve this problem, we use three functions:

1. **find** (a_{ij}, *wordlist*): Given an element a_{ij}, return all possible characters in wordlist that satisfy the puzzle.
2. **generate** (*list1*, *list2*, *list3*): Given three lists of characters, return all possible words.
3. **test** (*generated word*, *wordlist*): Given a generated word, test whether or not word in the given wordlist, return all generated words if they are in wordlist.

Algorithm Analysis: We now take a closer look into the efficiency of proposed algorithm. First of all, since we are using brute-force search, which exhaustively initializes all words in the given wordlist. Thus, we ensure that all solutions are covered, i.e the algorithm is completeness. Secondly, the **find** and **test** functions have $O(n)$ time complexity, where n is the number of words in wordlist.

Algorithm 2 Crossword Puzzle Solver

```
initialize $S$
for word in wordlist:
    $S(a_{00}, a_{01}, a_{11}) \leftarrow$ word
    for $i$ in 0 : 2
        $A_{1i}$ = find ($a_{0i}$, wordlist)
    $GA_1$ = generate ($A_{10}, A_{11}, A_{12}$)
    $A_1$ = test ($GA_1$, wordlist)
    if $A_1 = \emptyset$, break
    else for $a_1$ in $A_1$
        $S(a_{10}, a_{11}, a_{12}) \leftarrow a_1$
        for $i$ in 0 : 2
            $A_{2i}$ = find ($a_{1i}$, wordlist)
        $GA_2$ = generate ($A_{20}, A_{21}, A_{22}$)
        $A_2$ = test ($GA_2$, wordlist)
        if $A_2 = \emptyset$, break
        else for $a_2$ in $A_2$
            $S(a_{20}, a_{21}, a_{22}) \leftarrow a_2$
            return $S$
```

The **generate** has the time complexity of fixed $c = 26^3$, since it generates words from three lists having at most 26^3 elements. Hence, the time complexity of Algorithm 2 is given as follows:

$$T(n) \leq n(3n + 2c(3n + 2c)) \approx O(n^3)$$

Generally, the cardinal of A_{1i} and A_{2i} are at most 26 (number of letter in English alphabet), thus $|GA_1|$ and $|GA_2|$ are at most 26^3. The number of elements in A_1 and A_2 are otherwise depend on input wordlist, having at most n possible returns.
Algorithm Validation: To validate the proposed algorithm, we use a back-ward process. In contrast to solving the puzzle, we first create a crossword tables then generate the wordlist.

$$\begin{bmatrix} B & A & D \\ A & G & I \\ D & I & E \end{bmatrix} \begin{bmatrix} B & A & D \\ A & G & I \\ T & E & E \end{bmatrix} \begin{bmatrix} B & A & T \\ A & G & E \\ D & I & E \end{bmatrix} \begin{bmatrix} B & A & T \\ A & G & E \\ T & E & E \end{bmatrix}$$

Hence, we know the result beforehand. We use the following wordlist: ('bad', 'agi', 'tee', 'die', 'age', 'bat'), giving us exactly expected results under the algorithm.

Solution: In case of without repetition, there is no solution for the crossword puzzle. Every iteration broke at step testing GA_1. If we allow the repetition, we have the solution below:

$$A_1 = A_1^T = \begin{bmatrix} S & E & A \\ E & A & R \\ A & R & E \end{bmatrix}, \text{and } A_2 = A_2^T = \begin{bmatrix} S & E & A \\ E & A & R \\ A & R & T \end{bmatrix}$$

3 Planning with Certainty

Problem: LeBran finds a part-time job to pack products into a box of size $12.5 \times 7.5 \times 5.5$ (in inch). If he packs a product A of size $4 \times 3 \times 2$ into box, he can earn \$3. If he packs a product B of size $3 \times 2 \times 2$, he can earn \$1.5. If he packs a product C of ball with radius 1, he can earn \$1. Each box must contain at least one product A and one product B.
(a) How can LeBran earn most money?
(b) f the empty space must fill in soft medium to avoid instability, and each cubic inch of soft medium costs LeBran \$0.25. How can LeBran earn most money?
Problem Assessment: We first compare the efficiency of each product, in term of occupied volume and profit. Figure **??** shows that if we put 2 items B next to each other by surface 3×2, we will

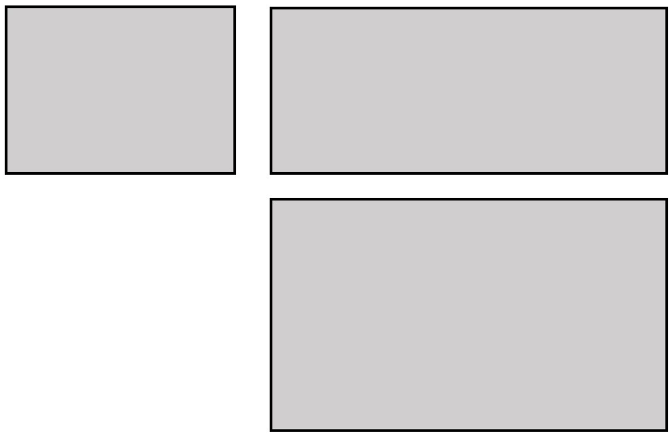


Figure 2: 3-D space can be represented as three 2-D planes

obtain exactly item A. Besides, stacking 6 balls of item C will result in 2 items A (append by surface 4×2). However, stacking balls will make leftover volume in the box. Thus we only attempt to fill the box with item A and B. Secondly, the given box is of size $12.5 \times 7.5 \times 5.5$, while the dimension of each item are integers, thus we can assume that the box is of item $12 \times 7 \times 5$. By simplify it, we can break down each dimension into integers, in which:

$$5 = 2 + 3, 7 = 3 + 4, 12 = 6 \times 2 = 4 \times 3 = 3 \times 4 \tag{1}$$

Planning strategy: We approach the problem by a backward process, that include 2 steps: (1) fill the box with the most number of items A and (2) replace one item A by two items B to satisfy the constrain. By this strategy, we greedy maximize the profit, since the profit of item A is the largest. Moreover, placing box item next to each other *will not* create empty space like in sphere objects, thus this is the solution for both (a) and (b).
Algorithm: Now we propose an algorithm to placing as most as possible item A into the box of size $12 \times 7 \times 5$. We can instead break the given box by three 2-D planes (Figure 2). The solution as in Figure [3, 4, 5, 6]
Solution: The total number of item A is: $7 + 6 + 3 = 16$. We can replace any item A by appending two items B by surface 3×2. The optimal profit found by proposed algorithm is $\$48$ for part **(a)**. When we fill the empty space with soft medium, the profit is: $48 - 0.25 \times (1 \times 6 \times 12) = 30$.

References

[1] Kwang-Cheng Chen, ”Artificial Intelligence in Wireless Robotics”

[2] Jon Kleinberg, Éva Tardos, ”Algorithm Design”

[3] Robert Sedgewick, ”Algorithms”

A Appendix

You may include other additional sections here.

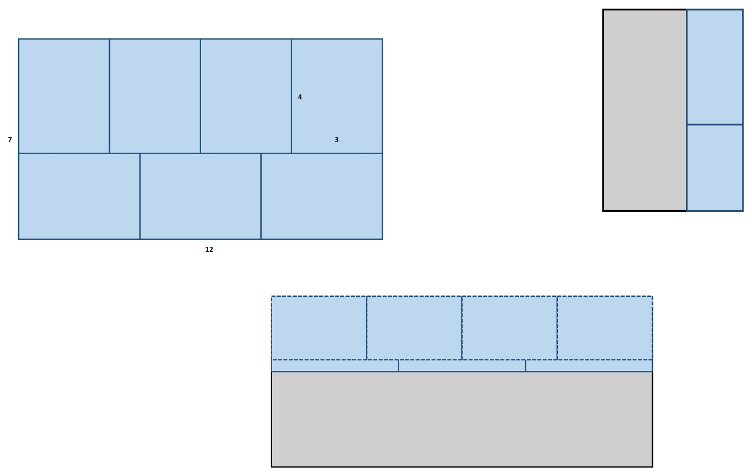


Figure 3: **Step 1:** We can arbitrarily choose one 2-D plane to start with. In this case, we start with 12×7 plane, resulting in fully filled space (top left panel). As a constrain of item dimension, the 2-D planes 7×5 and 12×5 will be filled automatically (2 remaining panel). We prioritize the case that leftover space can be gain represented as 3 rectangular planes. *(Dashed lines represents overlaid objects)*

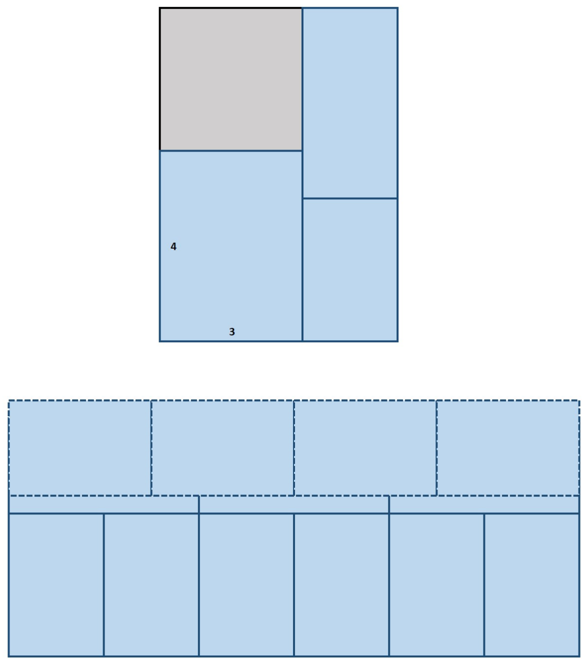


Figure 4: **Step 2:** Repeat step 1 on leftover space, now the original box is reduced to dimension of $12 \times 7 \times 3$. Again, arbitrarily fill one 2-D plane, then the remaining plane will automatically filled.

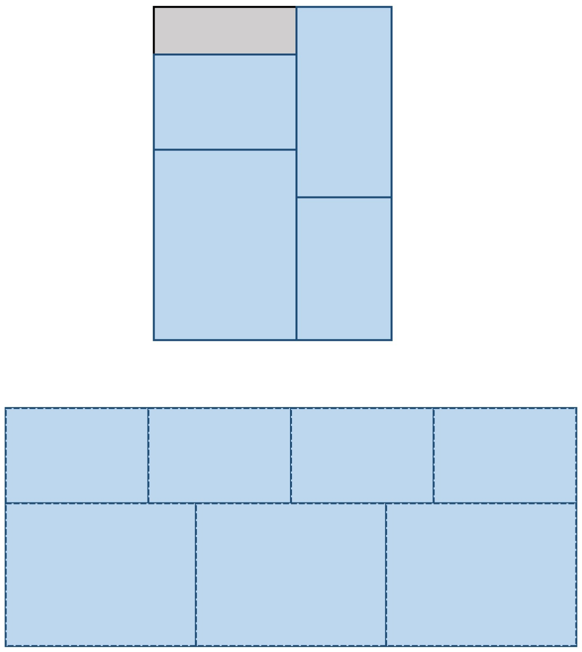


Figure 5: Repeat step 2 on space $12 \times 3 \times 3$

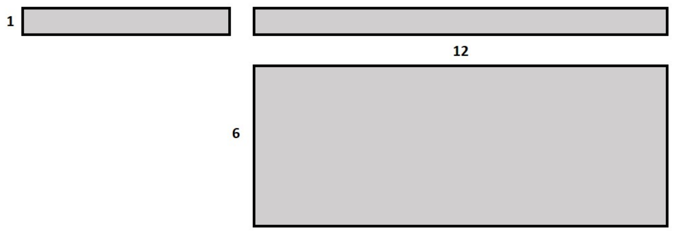


Figure 6: **Stopping criteria:** The algorithm will be terminated if one dimension reduced to 1, since we cannot further place any item. We form a solution by replace any item A by 2 item B appending by surface 3×2.