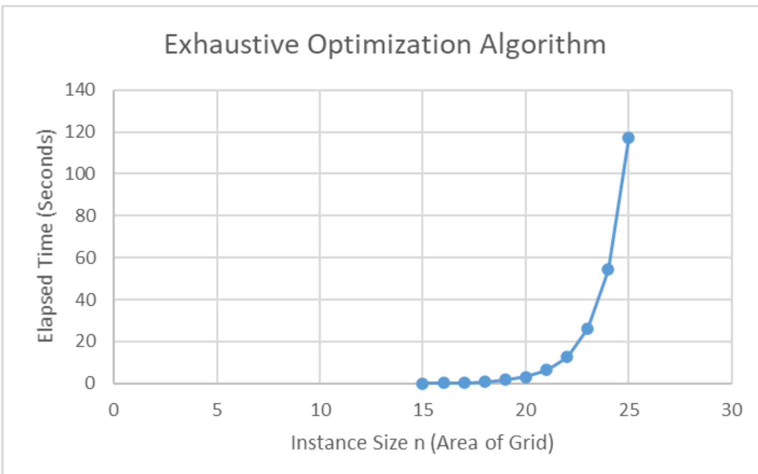
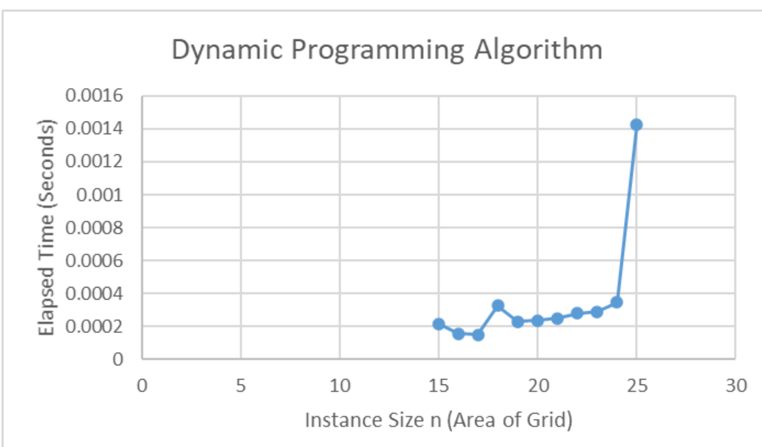
CPSC 335 Project 4

Names/Email:

Justin Bui: Justin_Bui12@csu.fullerton.edu
Benson Lee: blee71@csu.fullerton.edu

Scatterplots





Pseudocode

Exhaustive:

```
def crane_unloading_exhasutive (setting):
    assert(setting.rows() > 0)
    assert(setting.columns() > 0)

    max_steps = setting.rows() + setting.columns() - 2
    assert(max_steps < 64)
    best = None

     for steps = 1 to max_steps inclusive:
           for bits = 0 to (2^steps) - 1 inclusive:
               candidate = [start]
               valid = true
               for k = 0 to steps - 1 inclusive:
                     bit = (bit >> k) & 1

                        if (bit == 1):
                             if (candidate.is_step_valid(STEP_DIRECTION_EAST):
                                 candidate.add_step(STEP_DIRECTION_EAST):
                             else valid = false
                        else:
                              if (candidate.is_step_valid(STEP_DIRECTION_SOUTH):
                                  candidate.add_step(STEP_DIRECTION_SOUTH)
                              else valid = false

           endfor
           if (valid && (candidate.total_cranes() > best.total_cranes())):
               best = candidate
        endfor
    endfor
```

Dynamic:

```
crane_unloading_dyn_prog(setting):
    assert(setting.rows() > 0)
    assert (setting.columns > 0)

    A = (setting.rows(), vector<cell_type>(setting.columns()))
    A[0][0] = path(setting)
    assert(A[0][0].hash_value())

    for r = 0 to setting.rows() - 1:
        for c = 0 to setting.columns() - 1:
            if (setting.get(r, c) != CELL_BUILDING):
                from_above = None
                from_left = None

                if (r > 0 && A[r - 1][c].has_value()):
                    from_above = A[r -1][c]
                     if (from_above->is_step_valid(STEP_DIRECTION_SOUTH):
                          from_above->add_step(STEP_DIRECTION_SOUTH)

                if (c > 0 && A[r][c - 1].has_value()):
                    from_left = A[r][c - 1]
                    if (from_left->is_step_valid(STEP_DIRECTION_EAST)):
                          from_left->add_step(STEP_DIRECTION_EAST)

                if (from_above.has_value() && from_left.has_value()):
                    if (from_above->total_cranes() > from_left->total_cranes()):
                        A[r][c] = from_above
                    else: A[r][c] = from_left

               if (from_above.has_value() && !(from_left.has_value())):
                   A[r][c] = from_above

               if (from_left.has_value() && !(from_above.has_value())):
                   A[r][c] = from_left

            endif
        endfor
    endfor
```

```
// Post-processing to find maximum-crane path
best = A[0][0]
assert(best->has_value())
for r = 0 to setting.rows() - 1:
   for c = 0 to setting.columns() - 1:
      if (A[r][c].has_value() && A[r][c]->total_cranes() > (*best)->total_cranes()):
         best = &(A[r][c])

assert(best->has_value())
return **best
```

Step Count/Time Complexity Proofs

Exhaustive:

```
def crane_unloading_exhasutive (setting):
    assert(setting.rows() > 0)  (3)
    assert(setting.columns() > 0)  (3)

    max_steps = setting.rows() + setting.columns() - 2  (5)
    assert(max_steps < 64)  (2)
    best = None  (1)

     for steps = 1 to max_steps inclusive:
           for bits = 0 to (2^steps) - 1 inclusive:
              candidate = [start]  (1)
              valid = true  (1)
              for k = 0 to steps - 1 inclusive:
                      bit = (bit >> k) & 1  (3)

                      Entire if block below: 1 + max(1 + max(1, 1), 1 + max(1, 1)) = 1 + max(2, 2) = 1 + 2 = 3
                      if (bit == 1):   (1)
                         if (candidate.is_step_valid(STEP_DIRECTION_EAST):  (1)
                            candidate.add_step(STEP_DIRECTION_EAST):  (1)
                         else valid = false  (1)
                       else:
                          if (candidate.is_step_valid(STEP_DIRECTION_SOUTH):  (1)
                             candidate.add_step(STEP_DIRECTION_SOUTH)  (1)
                          else valid = false  (1)

               endfor
         if (valid && (candidate.total_cranes() > best.total_cranes())):  (4)
           best = candidate (1)
      endfor
    endfor
```

Step Counts:

Step Count = $3 + 3 + 5 + 2 + 1 + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}[(1 + 1 + \sum_{k=0}^{s-1}(3 + 3)) + 5]$

= $14 + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}[2 + \sum_{k=0}^{s-1}(6) + 5]$

= $14 + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}[2 + 6(s - 1 - 0 + 1) + 5]$

= $14 + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}[6s + 7]$

= $14 + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}(6s) + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}(7)$

= $14 + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}(6s) + \sum_{s=1}^{n}\sum_{b=0}^{2^s-1}(7)$

= $\sum_{s=1}^{n}\sum_{b=0}^{2^s-1}(6s)$ = $\sum_{s=1}^{n} 6s(2^s - 1 + 1)$ = $\sum_{s=1}^{n} 6s(2^s)$ = $6\sum_{s=1}^{n} s(2^s)$

= $6(1(2^1) + 2(2^2) + \cdots + n(2^n))$ = $12(1 - 2^n + 2^n n)$

= $\sum_{s=1}^{n}\sum_{b=0}^{2^s-1}(7)$ = $\sum_{s=1}^{n} 7(2^s - 1 - 0 + 1)$ = $\sum_{s=1}^{n} 7(2^s)$ = $7(2^1 + 2^2 + 2^3 + \cdots + 2^n)$

= $7[2(2^n - 1)]$ = $14(2^n - 1)]$

Putting it all together:

= $14 + \left(12(1 - 2^n + 2^n n)\right) + 14(2^n - 1)$

= $14 + 12 - 12(2^n) + 12(2^n)(n) + 14(2^n) - 14$

ANSWER = $12(2^n)(n) + 2(2^n) + 12$

Proof Using Limit Theorem:

f(n) belongs in O(g(n)) if $\lim_{n\to\infty}\frac{f(n)}{g(n)} = L \text{ where } L \geq 0 \text{ and is constant.}$

Let f(n) = $12(2^n)(n) + 2(2^n) + 12$, g(n) = $n^2(2^n)$

$$\lim_{n\to\infty}\frac{12(2^n)(n) + 2(2^n) + 12}{(n^2)(2^n)} = \lim_{n\to\infty}\frac{12\cancel{(2^n)}\cancel{n}}{(n^{\cancel{2}})\cancel{(2^n)}} + \frac{2\cancel{(2^n)}}{(n^2)\cancel{(2^n)}} + \frac{12}{(n^2)(2^n)}$$

= $\lim_{n\to\infty}\frac{12}{n} + \frac{2}{n^2} + \frac{12}{(n^2)(2^n)} = 0$

Therefore, $12(2^n)(n) + 2(2^n) + 12$ belongs to $(n^2)(2^n)$. ■

Dynamic:

```
crane_unloading_dyn_prog(setting):
  assert(setting.rows() > 0)  (3)
  assert (setting.columns > 0)  (3)

  A = (setting.rows(), vector<cell_type>(setting.columns()))  (3)
  A[0][0] = path(setting)  (2)
  assert(A[0][0].hash_value())  (2)

  for r = 0 to setting.rows() - 1:  ((n - 1) - 0 + 1) = n
    for c = 0 to setting.columns() - 1:  (n - 1 - 0 + 1) = n
      if (setting.get(r, c) != CELL_BUILDING): (2)
        from_above = None  (1)
        from_left = None  (1)

        if (r > 0 && A[r - 1][c].has_value()):  (4)
          from_above = A[r -1][c]  (2)
            if (from_above->is_step_valid(STEP_DIRECTION_SOUTH): (1)
              from_above->add_step(STEP_DIRECTION_SOUTH) (1)
                (THIS BLOCK = 4 + 2 + 1 + 1 = 8)

        if (c > 0 && A[r][c - 1].has_value()): (4)
          from_left = A[r][c - 1]  (2)
            if (from_left->is_step_valid(STEP_DIRECTION_EAST)): (1)
              from_left->add_step(STEP_DIRECTION_EAST)        (1)
                (THIS BLOCK = 4 + 2 + 1 + 1 = 8)

        if (from_above.has_value() && from_left.has_value()): (3)
          if (from_above->total_cranes() > from_left->total_cranes()): (3)
            A[r][c] = from_above (1)
          else: A[r][c] = from_left (1)
                (THIS BLOCK = 3 + max(3 + max(1, 1), 0) = 3 + max(4, 0) = 3 + 4 = 7)

        if (from_above.has_value() && !(from_left.has_value())): (4)
          A[r][c] = from_above (1)

        if (from_left.has_value() && !(from_above.has_value())): (4)
          A[r][c] = from_left (1)

      endif
    endfor
  endfor

  // Post-processing to find maximum-crane path
  best = A[0][0]  (1)
  assert(best->has_value()) (2)
  for r = 0 to setting.rows() - 1:  (n - 1 - 0 + 1) = n
    for c = 0 to setting.columns() - 1:  (n - 1 - 0 + 1) = n
      if (A[r][c].has_value() && A[r][c]->total_cranes() > (*best)->total_cranes()): (4)
        best = &(A[r][c])  (1)

  assert(best->has_value())  (2)
  return **best  (0)
```

Step Count:

sc = 3 + 3 + 3 + 2 + 2 + n [n * (2 + max(1 + 1 + 8 + 8 + 7 + 5 + 5, 0)))] + 3 + $5n^2$

sc = 16 + n [n * (2 + 35))] + $5n^2$

sc = 16 + $37n^2$ + $5n^2$

sc = 16 + $42n^2$

Step Count = $42n^2 + 16$

Proof Using Limit Theorem:

$f(n)$ belongs in $O(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = L$, where $L \geq 0$ and is constant.

Let $f(n) = 42n^2 + 16$, $g(n) = n^3$.

$$\lim_{n \to \infty} \frac{42n^2 + 16}{n^3} = \lim_{n \to \infty} \frac{42\cancel{n^2}}{n^{\cancel{3}}} + \frac{16}{n^3} = \lim_{n \to \infty} \frac{42}{n} + \frac{16}{n^3} = \frac{42}{\infty} + \frac{16}{\infty^3} = 0$$

Question Responses

a. **Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?**

Yes, there is a noticeable difference between the two algorithms. Dynamic programming algorithm is about 80,000 times faster than the exhaustive optimization algorithm. For dynamic programming (as seen in the scatterplot), each elapsed time to execute takes less than a second to execute as the instance size increases, while the elapsed time for exhaustive optimization exponentially increases, reaching beyond a minute to execute as the instance size increases. This does not surprise us at all.

b. **Are your empirical analyses consistent with your mathematical analyses? Justify your answer.**

Yes, our empirical analyses are consistent with our mathematical analyses. When computing the step count for the exhaustive algorithm, we've proven that it belongs to O(n^2*2^n), which makes this algorithm extremely slow as being in exponential time. On the other hand, when computing the step count for the dynamic programming algorithm, we've noticed that it belongs to O(n^3), which makes this algorithm more efficient as it is in polynomial time.

c. **Is this evidence consistent or inconsistent with the hypothesis? Justify your answer.**

The evidence is consistent with the hypothesis. When looking at the scatterplots, we can see that dynamic programming can handle larger instances within less than seconds compared to dynamic. We also see that dynamic belongs to a faster time complexity class being polynomial compared to dynamic being exponential.