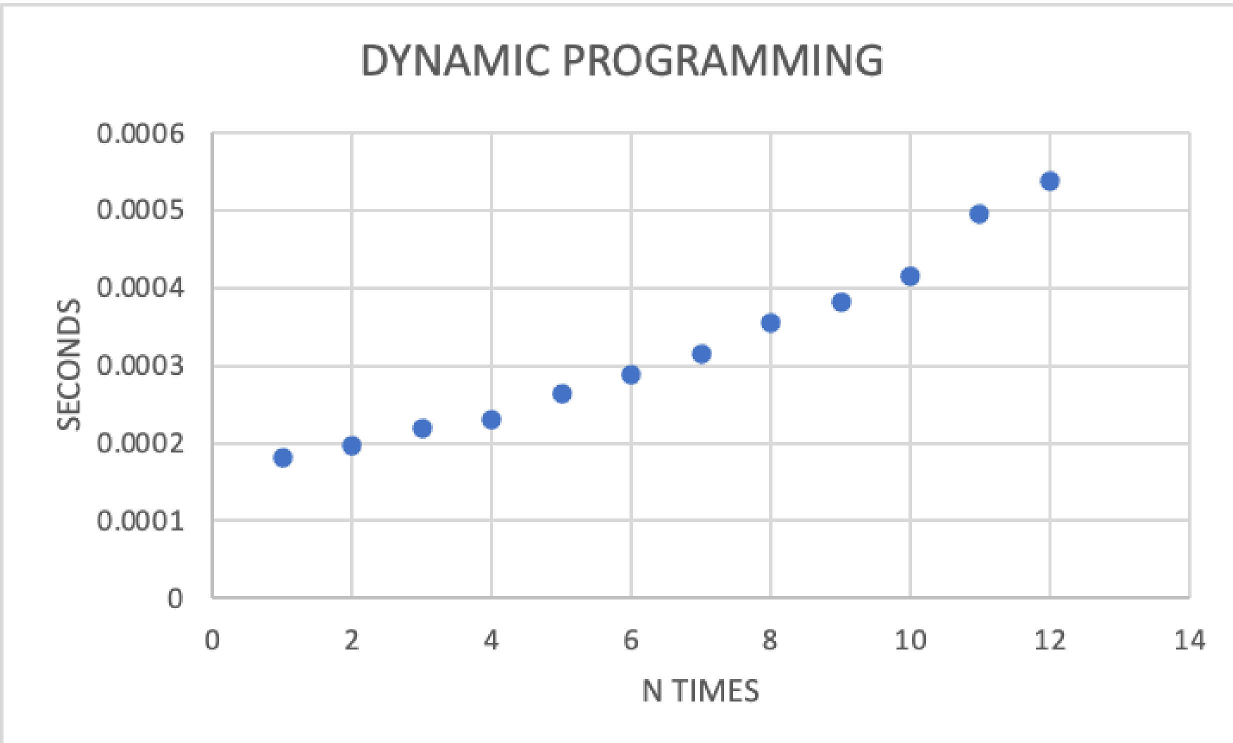
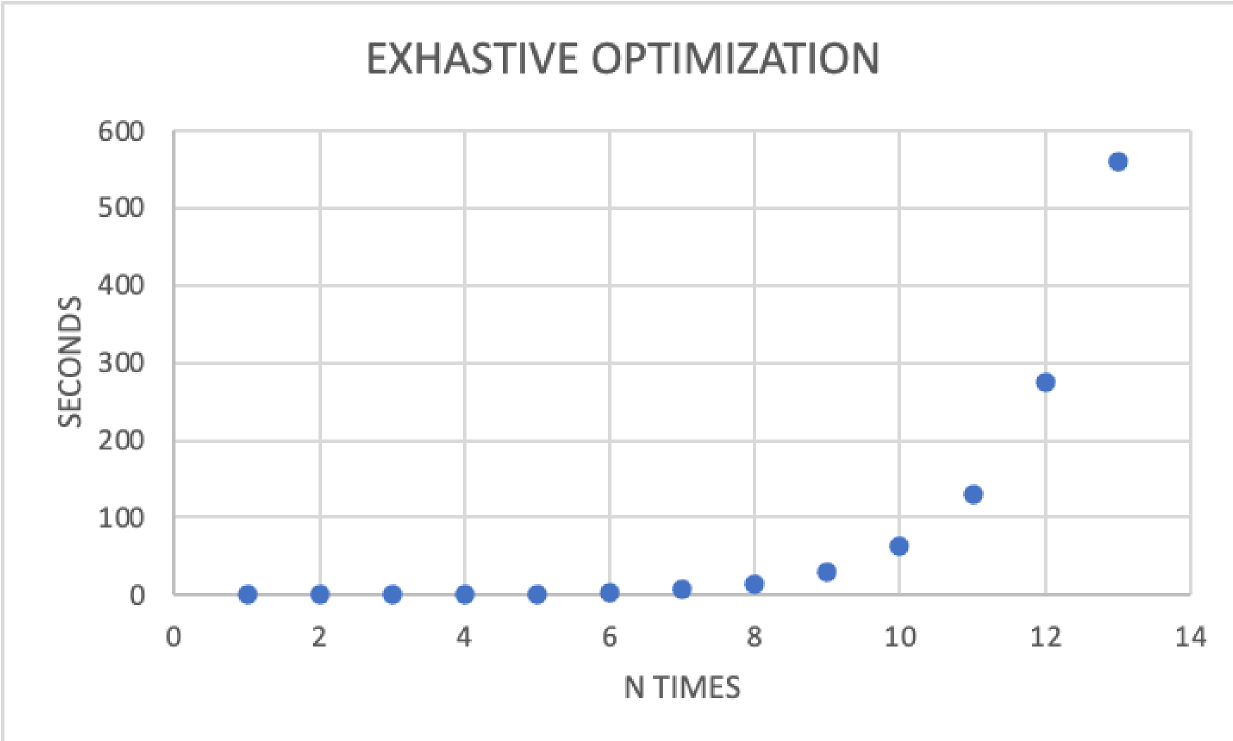
CPSC 335 - Project 4 PDF REPORT

Ali Tahami
atahami3@csu.fullerton.edu

Hamid Suha
hsuha@csu.fullerton.edu




EXHASTIVE OPTIMIZATION
SECONDS
N TIMES
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14

EXHAUSTIVE OPTIMIZATION ALGORITHM

```
for (size_t steps = 1; steps <= max_steps; ++steps)
  {
   uint64_t mask = uint64_t(1) << steps;
   for (uint64_t bits = 0; bits < mask; ++bits)
   {

    path candidate(setting);

    for( uint64_t k = 0; k < steps; ++k)
    {
     int bit;
     bit = (bits >> k) & 1;

     if(bit == 1)
     {
        if(candidate.is_step_valid(STEP_DIRECTION_EAST))
        {
         candidate.add_step(STEP_DIRECTION_EAST);
        }
     }
     else
     {
        if(candidate.is_step_valid(STEP_DIRECTION_SOUTH))
         {
          candidate.add_step(STEP_DIRECTION_SOUTH);
         }
     }


    }

    if (candidate.total_cranes() > best.total_cranes())
    {
     best = candidate;
    }
   }
  }
  return best;
```

DYNAMIC PROGRAMMING ALGORITHM

```
for (coordinate r = 0; r < setting.rows(); ++r)
 {
  for (coordinate c = 0; c < setting.columns(); ++c)
  {
   if (setting.get(r, c) != CELL_BUILDING)
     {
      std::optional<path> from_above;
      std::optional<path> from_left;
      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;
         }
       }
      else if(from_above.has_value())
         {
          A[r][c] = from_above;
         }
      else if(from_left.has_value())
         {
          A[r][c] = from_left;
         }
     }
  }
 }

 cell_type* best = &(A[0][0]);
 assert(best->has_value());
 for (coordinate r = 0; r < setting.rows(); ++r)
 {

   for (coordinate c = 0; c < setting.columns(); ++c)
     {
      if (A[r][c].has_value() && A[r][c]->total_cranes() > (*best)->total_cranes())
       {
        best = &(A[r][c]);
       }
     }
 }

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

EXHAUSTIVE OPTIMIZATION STEP COUNT AND PROOF

```
for (size_t steps = 1; steps <= max_steps; ++steps)
 {
  uint64_t mask = uint64_t(1) << steps;
  for (uint64_t bits = 0; bits < mask; ++bits)
  {

   path candidate(setting); // 0

   for( uint64_t k = 0; k < steps; ++k) // i
   {
    int bit;
    bit = (bits >> k) & 1; // 3

    if(bit == 1) // 1+max(2,2) = 3
    {
      if(candidate.is_step_valid(STEP_DIRECTION_EAST)) // 1+max(1,0) = 2
      {
       candidate.add_step(STEP_DIRECTION_EAST);
      }
    }
    else
    {
      if(candidate.is_step_valid(STEP_DIRECTION_SOUTH)) // 1+max(1,0)= 2
       {
        candidate.add_step(STEP_DIRECTION_SOUTH);
       }
    }


   }

   if (candidate.total_cranes() > best.total_cranes()) // 1+max(1,0)= 2
   {
    best = candidate;
   }
  }
 }
 return best; // 0
```

$\sum_{i=1}^{n+m-2} \sum_{j=0}^{2^i}$

$3+3+2=8$

$$\sum_{i=1}^{n+m-2} \sum_{j=0}^{2^i} 8i$$

$$\sum_{i=1}^{n+m-2} \sum_{j=0}^{2^i} 8i$$

$$\sum_{i=1}^{n+m-2} 8i\left(2^i - 0 + 1\right)$$

$$= \sum_{i=1}^{n+m-2} 8i\left(2^i\right) + 8i$$

$$= \underbrace{\sum_{i=1}^{n+m-2} 8i\left(2^i\right)} + \underbrace{\sum_{i=1}^{n+m-2} 8i}$$

$$\sum_{i=1} 8i\left(2^i\right) = 8(1)\left(2^1\right) + 8(2)\left(2^2\right) + 8(3)\left(3^3\right) + \ldots + 8$$

$$= 4\left(m\,2^{m+n} - 3\left(2^{m+n}\right) + n\left(2^{m+n}\right) + 4\right)$$

$$\sum_{i=1}^{n+m-2} 8i = 4(m+n-2)(m+n-1)$$

$$4m^2 + 8mn - 12m + 4n^2 - 12n + 8$$

$$= 4\left(m\,2^{m+n} - 3\left(2^{m+n}\right) + n\left(2^{m+n}\right) + 4\right) + 4m^2 + 8mn - 12m + 4n^2 - 12n + 8$$

Tried getting step count another way

```
for steps=1 to max_steps // 2^n
    mask = << steps //2
    for bits =0 to mask-1 // n
        for k=0 to steps-1 // n
            if bit
                add_step
        if candidate > best
            best= candidate
return best
```

$sc = n(2+(2^n(n)+1))$

$sc = n(2+(2^n \cdot n+1))$

$sc = n(3+2^n \cdot n)$

$sc = \boxed{3n+n^2 \cdot 2^n}$

$O(n^2 \cdot 2^n)$

Proof: $\lim_{n\to\infty} = \dfrac{3n+n^2 \cdot 2^n}{n^2 \cdot 2^n}$

$\lim_{n\to\infty} = \dfrac{3+2n\ln(2)2^n}{2n\ln(2)2^n}$

$\lim_{n\to\infty} = \dfrac{\ln^2(2)2^n}{\ln^2(2)2^n}$

$= 1$ ✓ does belong

DYNAMIC PROGRAMMING STEP COUNT AND PROOF

```
for (coordinate r = 0; r < setting.rows(); ++r) n                21n^2
 {
  for (coordinate c = 0; c < setting.columns(); ++c) n
  {
   if (setting.get(r, c) != CELL_BUILDING) 2+max(19,0) = 21
    {
     std::optional<path> from_above; 0
     std::optional<path> from_left; 0
     if(r > 0 && A[r-1][c].has_value()) 3+max(3,0)=6
      {
       from_above = A[r-1][c]; 1
       if(from_above->is_step_valid(STEP_DIRECTION_SOUTH)) // 1+max(1,0)=2
        {
         from_above->add_step(STEP_DIRECTION_SOUTH);
        }
      }
     if(c > 0 && A[r][c-1].has_value()) 3+max(3,0)=6
      {
       from_left = A[r][c-1]; //1
       if(from_left->is_step_valid(STEP_DIRECTION_EAST)) // 1+max(1,0)=2
        {
         from_left->add_step(STEP_DIRECTION_EAST);
        }
      }
     if(from_above.has_value() && from_left.has_value()) 3+max(4,3)=7
      {
       if(from_above->total_cranes() > from_left->total_cranes()) 3+max(1,1)= 4
        {
         A[r][c] = from_above;
        }
       else
        {
         A[r][c] = from_left;
        }
      }
     else if(from_above.has_value()) 1+max(1,2) = 3
        {
         A[r][c] = from_above;
        }
     else if(from_left.has_value()) 1+max(1,0)= 2
        {
         A[r][c] = from_left;
        }
    }
  }
 }

 cell_type* best = &(A[0][0]); 1
 assert(best->has_value()); 1                                   6n^2+3
 for (coordinate r = 0; r < setting.rows(); ++r) n
 {

   for (coordinate c = 0; c < setting.columns(); ++c) n
    {
     if (A[r][c].has_value() && A[r][c]->total_cranes() > (*best)->total_cranes()) 5+max(1,0)=6
      {
       best = &(A[r][c]);
      }
    }
 }

 assert(best->has_value()); //1
 return **best; //0
```

PROOF

$$SC = 6n^2 + 3 + 21n^2$$

$$SC = \boxed{27n^2 + 3}$$

$$O(n^2)$$

$$\lim_{n \to \infty} \frac{27n^2 + 3}{n^2}$$

$$\lim_{n \to \infty} \frac{54n}{2n}$$

$$\lim_{n \to \infty} \frac{54}{2} = 27$$

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?** The two algorithms, exhaustive and dynamic, have a distinct difference. The dynamic algorithm is more efficient. The results were not that surprising; we expected exhaustive to be slower than dynamic. For dynamic programming, each time executed took less than a second, while exhaustive optimization execution times increased exponentially as instance size increased.

b. **Are your empirical analyses consistent with your mathematical analyses? Justify your answer.** Yes, our empirical analysis is consistent with our mathematical analysis. Because we concluded our runtime for the dynamic algorithm to be $O(n^2)$ while our exhaustive was $O(n^2 2^n)$. So exhaustive was much slower. The difference in our empirical analysis was also much slower which makes both algorithms consistent.

c. **Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.** Yes, Polynomial-time dynamic programming algorithms are a lot more efficient than exponential-time exhaustive optimization algorithms that solve the exact same problem.