Memory Utilization

Paul Franklin, Yangqi Su, Yaying Shi

1.7 Memory utilization

Input: A disk of size D. A set F of n files of size $f_1, \ldots, f_n$.
Output: A subset $S \subseteq F$ of the files that can fit on the disk.
Metric: Maximize the number of file in the disk.
(Bonus question: adapt the algorithm for knapsack, why isn't it optimal?)

Presentation of the problem

- Input:

  For any n files set, we have size set F for those files.

  F = $\{f_i | \ where \ i \ from \ 1 \ to \ n\} = \{f_1, f_2, f_3, \dots, f_n\}$

  You have a capacity of D which is a size limit.

- Output:

  A subset of F

  S = $\{s_i | \ where \ i \in S\} = \{s_1, s_2, s_3, \dots, s_k\}$

- Metric

  Max(k)

Important of this problem

- Same as Knapsack problem but all the object value is same.
- Any arrangement and schedule problem with a threshold
  1. If you have an exam tomorrow, how could you select the most value part in limited time.
  2. How could you schedule your travel plan with limited money?
  3. Data download management
  4. Cache select

Properties

- Strong Dominance property

  For question we have all optimal solutions S^* = (k, s_1, … , s_{k-1}, s_k) that D - $\sum_{i \in k} s_i < s = \{s \in F \cap \overline{S^*}\}$

- Weak Dominance property

  There is an optimal solution that all the elements is in increased order in F. That's to say, S^* is first k elements of sorted F.

Proofs

- Strong Proofs: Assume S^* = (k, s_1, … , s_k) is an optimal solution of problem I = (D , f_1 , … , f_n) of k files, but D - $\sum_{i \in k} s_i$ >= s {s | s ∈ F ∩ $\overline{S^*}$}. Then, we can construct a valid solution S = (k+1, s_1, … , s_k, s_i) for problem I where $\{s_i \mid (s_i \in F \cap \overline{S^*}) and \sum_{i \in k+1} s_i < D\}$ of k+1 files. It's conflicts with S^* is optimum for I .
- Weak Proofs: For any solution S = (i, s_1, … , s_i) , we can constructor a solution S' = (k',s'_1, s'_2, … , s'_k) by replace each elements in the solution by one or more smaller elements in F, which can guarantee k <= k'. That would also be true for an optimal solution.

Greedy Algorithm

- Algorithm

```
for i in range(len(F)):
        for j in range(len(F)-1)
        find smallest;
        sum += smallest;
        if (sum > D):
                break
return index
```

- Complexity:

$$O(n * n) = O(n^2)$$

Complexity and optimized

- Optimized:
  1. Sort F to F’
  2. for item in range(len(F’)):

     sum += item;

     if (sum > D):

     break

     return index
- Complexity

$$O(n * logn) + O(n) = O(n * logn)$$

Adapted Algorithm for knapsack

- Problem
  - Input:
    For any n object set, we have size set S and value set V for those objects.
    S = $\{s_i | \ where \ i \ from \ 1 \ to \ n\} = \{s_1, s_2, s_3, \dots, s_n\}$
    V = $\{v_i | \ where \ i \ from \ 1 \ to \ n\} = \{v_1, v_2, v_3, \dots, v_n\}$
    You have a capacity of C which is a size limit.
  - Output:
    A subset of S
    T = $\{s_i | \ where \ i \in$ T$\}$
  - Metric
    Max$\sum_{i \in T} v_i$
- Algorithm

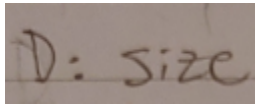
```
Sort S to S'
for item in range(len(S')):
        sum += item;
        val += V[I];
        if (sum > V):
                        break
    return val
```

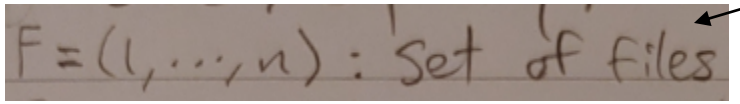
Why is not optimal in knapsack?

Memory Utilization is a special case in knapsack. When the smaller size object has same or great value than larger one, the knapsack is same as the Memory Utilization. But in knapsack problem, size is not the only threshold for question, the value is another hidden threshold. You must satisfied both of them. In Memory Utilization, when you satisfied one threshold, another is automatically satisfied.

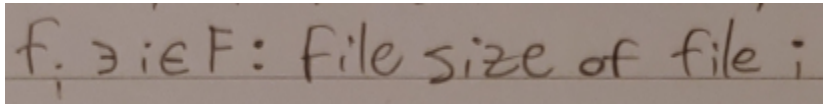
Presentation of the problem

Input

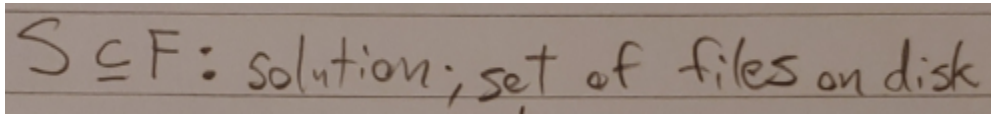




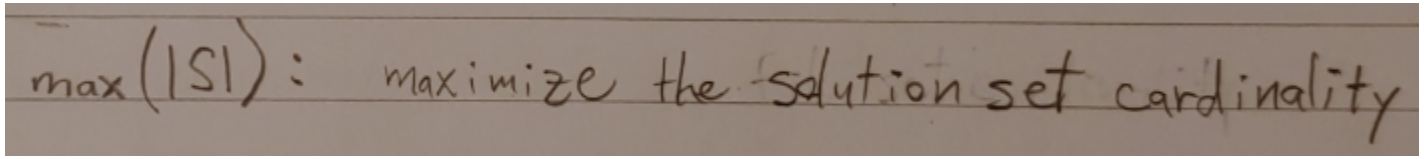
This definition changes to vector notation.



Output

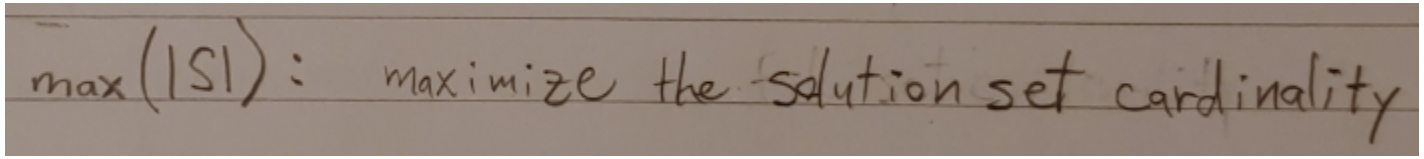


Metric

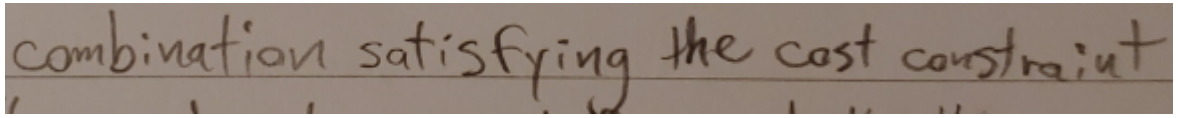


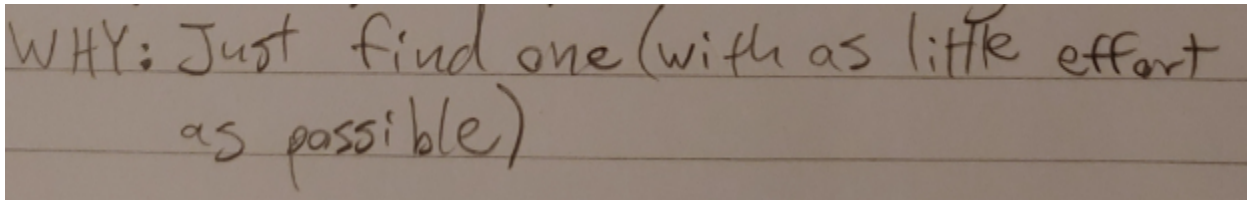
Without a Dominance Property

An optimal solution to the problem or any subproblem of the problem has the properties:



and





Note: It is impossible to compute what the value equivalent to the metric is until all combinations satisfying the cost constraint have been computed. Alternatively, the metric is a function of the set of ALL combinations satisfying the cost constraint. Also, the metric is a function of information that has not been obtained at any nonfinal, local decision.

(Weak) Dominance Property

Dominance Rule: eliminating uninteresting solutions, or selecting interesting solutions.

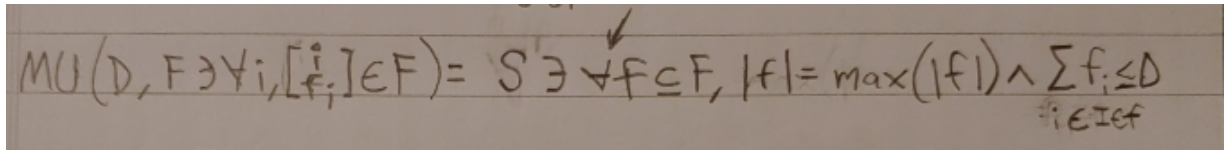
$S' = \{(1,1,1,1), (1,1,1,2)\}$

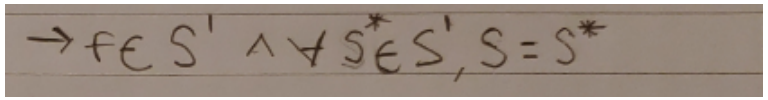
IF dominance rule selects minimum size, then $S'' = \{(1,1,1,1)\}$.

- IF, $\forall s \in S'$, $s \in S''$, THEN "Strong".

Formula

Parallel Computing





Formula (Dominance Property)

Dominance Rule (1), weak

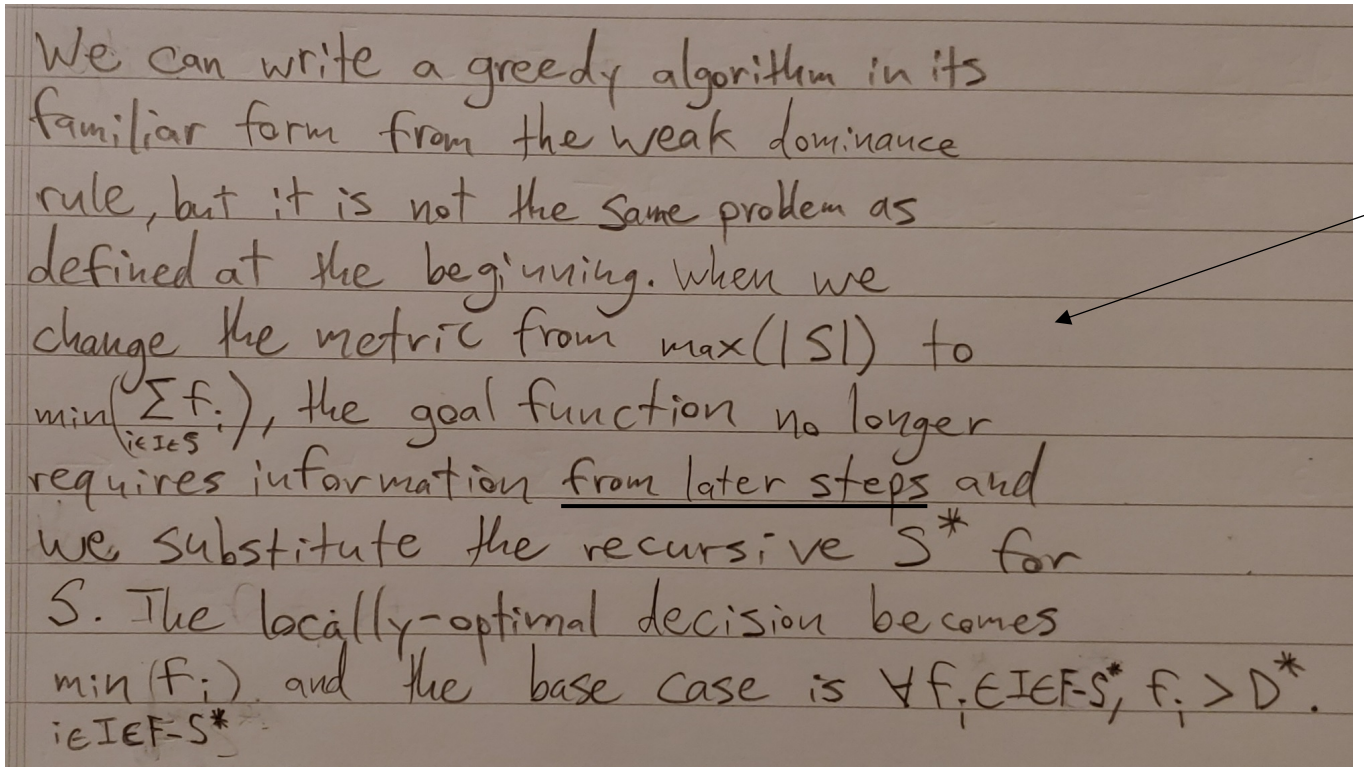
$$MU(D, F \ni \forall i, [\tfrac{i}{f_i}] \in F) = S' \ni \forall f \subseteq F, |f| = \max(|f|) \wedge \sum_{i \in I \in f} f_i \leq D$$

$$\rightarrow f \in S' \wedge \forall f \in S', \sum_{i \in I \in f} f_i = \min\left(\sum_{i \in I \in f} f_i\right) \rightarrow f \in S'' \wedge \forall S^* \in S'',$$

$$S = S^*$$

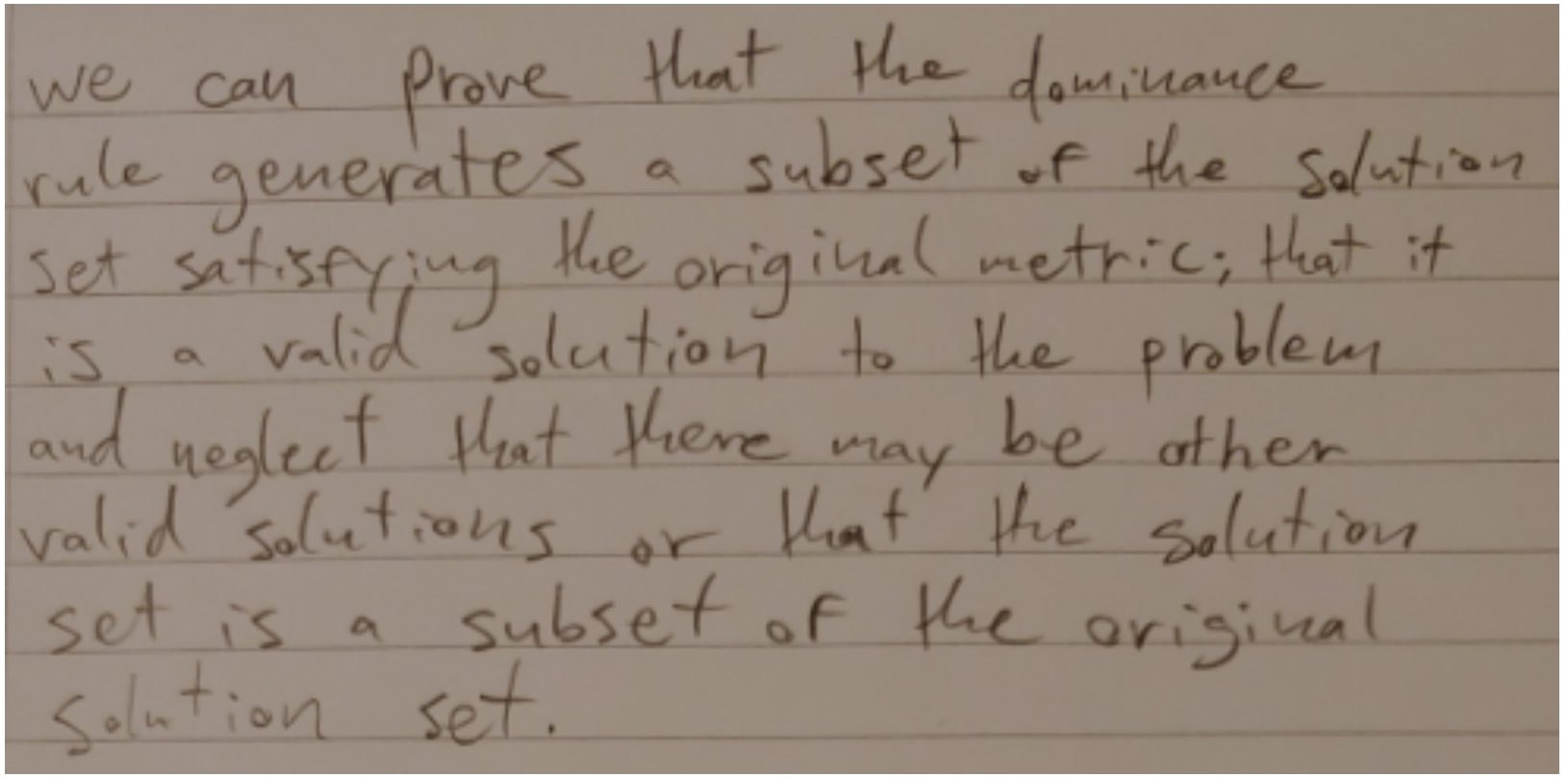
Alternatively, parallel computing distribution is a function of the first element. However, if our intent is not to implement a dynamic programming algorithm, there is a contradiction.

Formulas



"Globally Optimal" Metric

Proof(s)


we can prove that the dominance
rule generates a subset of the solution
set satisfying the original metric; that it
is a valid solution to the problem
and neglect that there may be other
valid solutions or that the solution
set is a subset of the original
solution set.

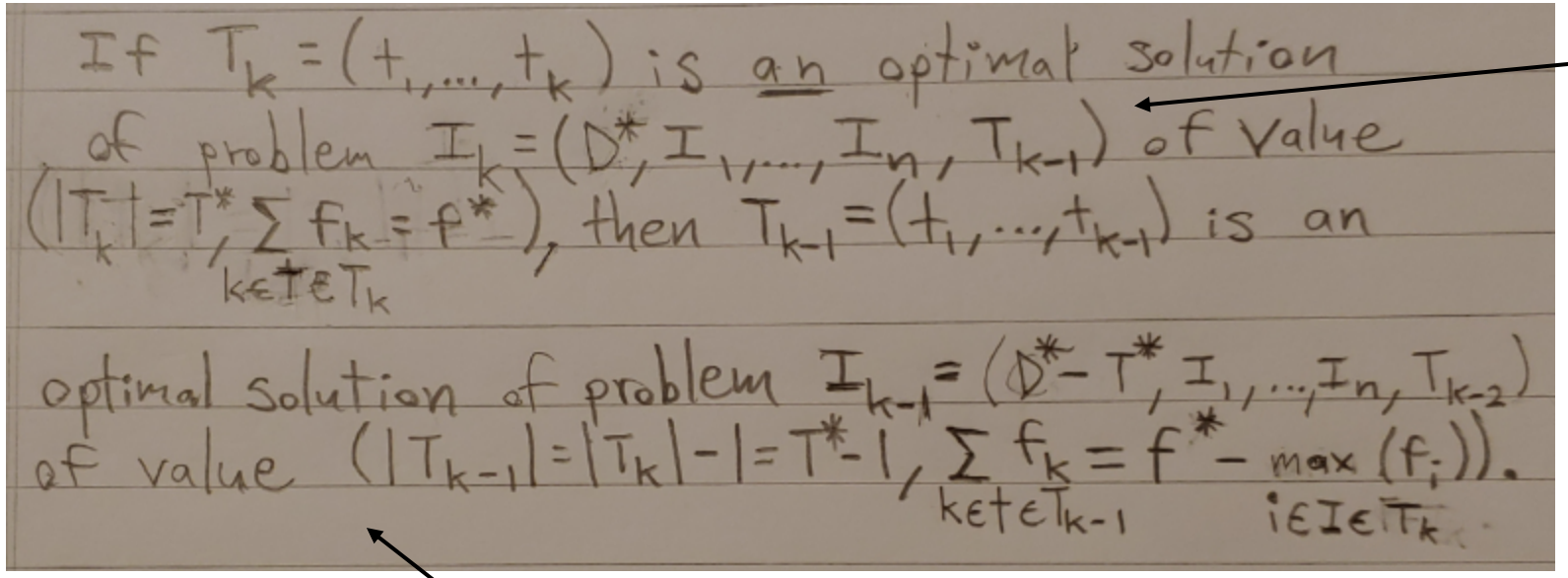
Proof(s) - Optimal Substructure

Let $T_k \subseteq S$.

Either assume all elements have distinct f_i, or that an arbitrary element or an element minimizing computational time complexity is selected from the extremas.

$$\max_{i \in I \in T_k}(f_i) = \min_{i \in I \in F - T_{k-1}}(f_i)$$

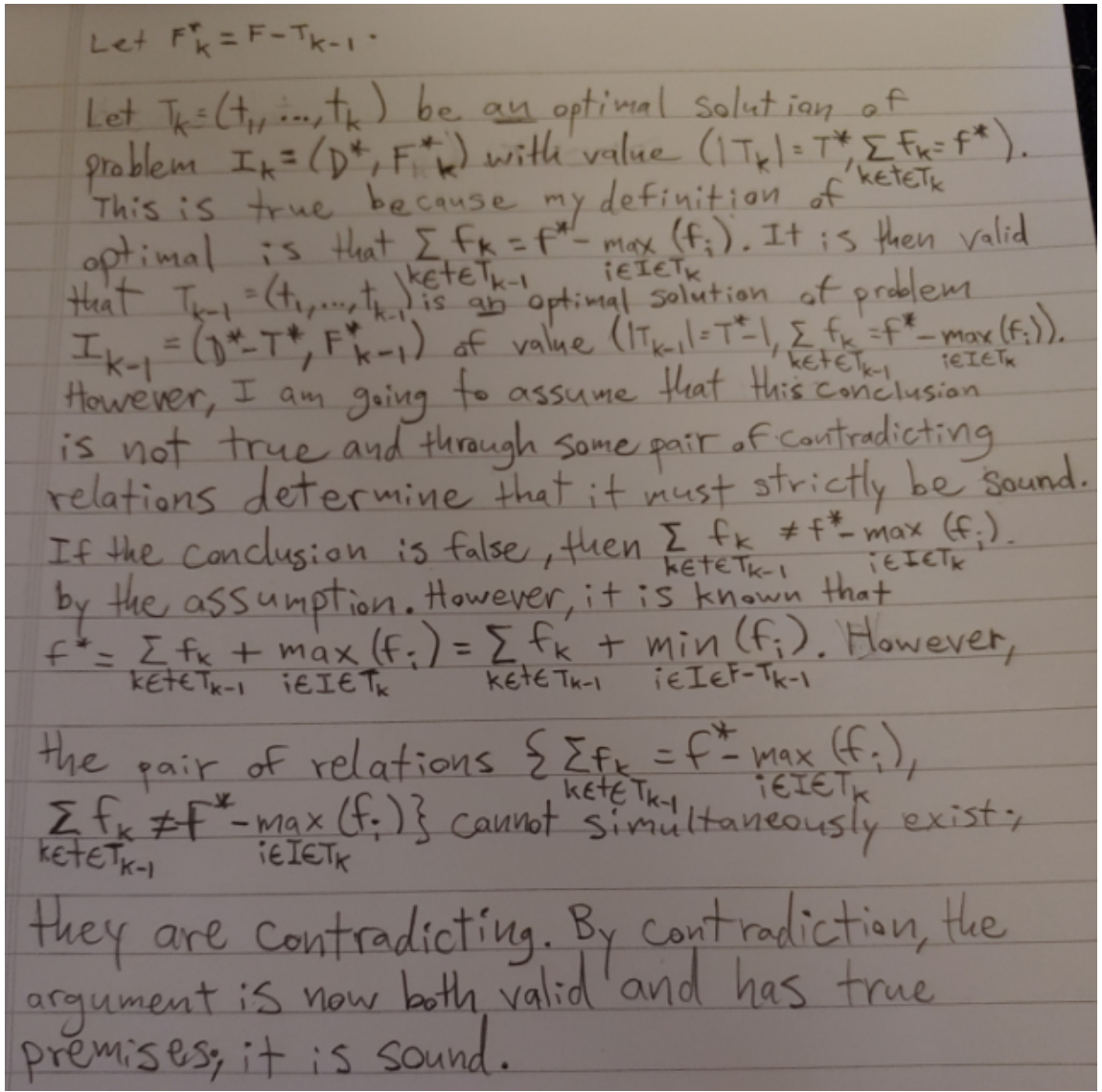
Proof(s) - Optimal Substructure



This could be further edited to say that the set of all vectors in any instance of input is represented by the variable F*, where $F^*_k = F - T_{k-1}$.

This recursion achieves the dominance rule, while also computing the metric, and this is how I've chosen to define what is optimal with respect to the problem. I go on to prove that the solutions generated by the dominance rule are always a subset of the solutions generated by the metric.

Proof(s) – Optimal Substructure



Let $F^*_k = F - T_{k-1}$.

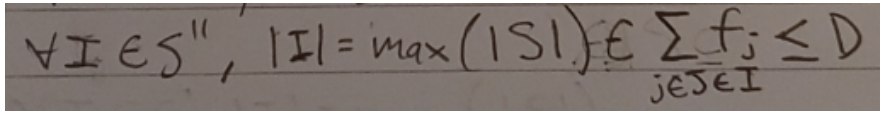
Let $T_k = (t_1, \ldots, t_k)$ be an optimal solution of problem $I_k = (D^*, F^*_k)$ with value $(|T_k| = T^*, \sum_{k \in t \in T_k} f_k = f^*)$. This is true because my definition of optimal is that $\sum_{k \in t \in T_{k-1}} f_k = f^* - \max_{i \in I \in T_k}(f_i)$. It is then valid that $T_{k-1} = (t_1, \ldots, t_{k-1})$ is an optimal solution of problem $I_{k-1} = (D^* - T^*, F^*_{k-1})$ of value $(|T_{k-1}| = T^* - 1, \sum_{k \in t \in T_{k-1}} f_k = f^* - \max_{i \in I \in T_k}(f_i))$. However, I am going to assume that this conclusion is not true and through some pair of contradicting relations determine that it must strictly be sound. If the conclusion is false, then $\sum_{k \in t \in T_{k-1}} f_k \neq f^* - \max_{i \in I \in T_k}(f_i)$ by the assumption. However, it is known that $f^* = \sum_{k \in t \in T_{k-1}} f_k + \max_{i \in I \in T_k}(f_i) = \sum_{k \in t \in T_{k-1}} f_k + \min_{i \in I \in F - T_{k-1}}(f_i)$. However, the pair of relations $\{\sum_{k \in t \in T_{k-1}} f_k = f^* - \max_{i \in I \in T_k}(f_i), \sum_{k \in t \in T_{k-1}} f_k \neq f^* - \max_{i \in I \in T_k}(f_i)\}$ cannot simultaneously exist; they are contradicting. By contradiction, the argument is now both valid and has true premises; it is sound.

Proof(s) – Dominance Property is Weak

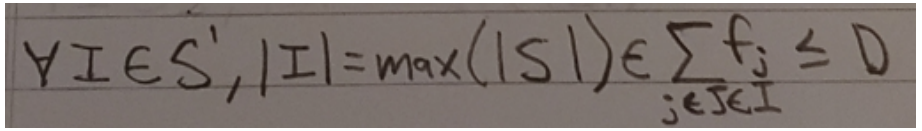
The dominance property generalizes to the metric; the solutions generated by the dominance property are a subset of the solutions generated by the metric.

Note: This is not a proof that the dominance rule is globally optimal. It is a proof that the dominance rule generates a subset of the solutions that satisfy the original metric. Now, it is unnecessary to compute all possible solutions to determine what the value of metric is.

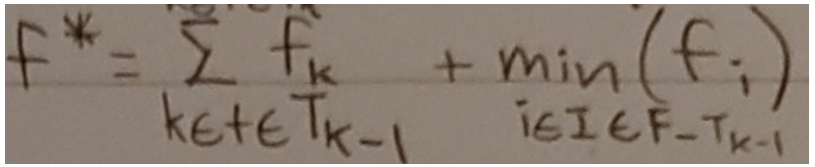
Proof(s) – Dominance Property is Weak



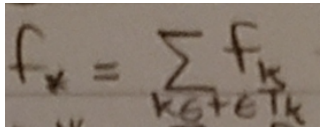
Equivalence (1)
The solutions without the dominance rule satisfy the metric and the capacity constraint.



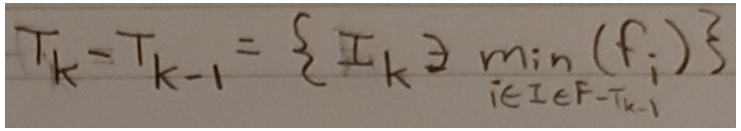
Equivalence (2)
The solutions with the dominance rule satisfy the metric and the capacity constraint.



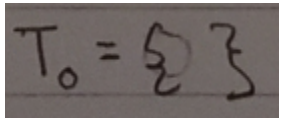
Equivalence (3)
Refer to the proof of optimal substructure.



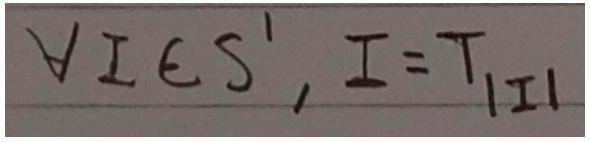
Equivalence (4)
Refer to the proof of optimal substructure.



Equivalence (5)
By Algebra on {3, 4}.

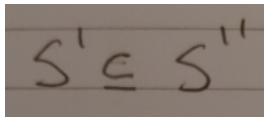


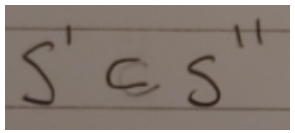
Equivalence (6)
Definition of base case.

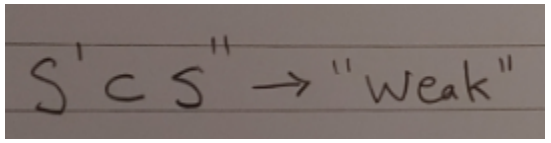


Equivalence (7)
From equivalence rules {5, 6}, derived from the definition of the dominance rule.

Proof(s) – Dominance Property is Weak



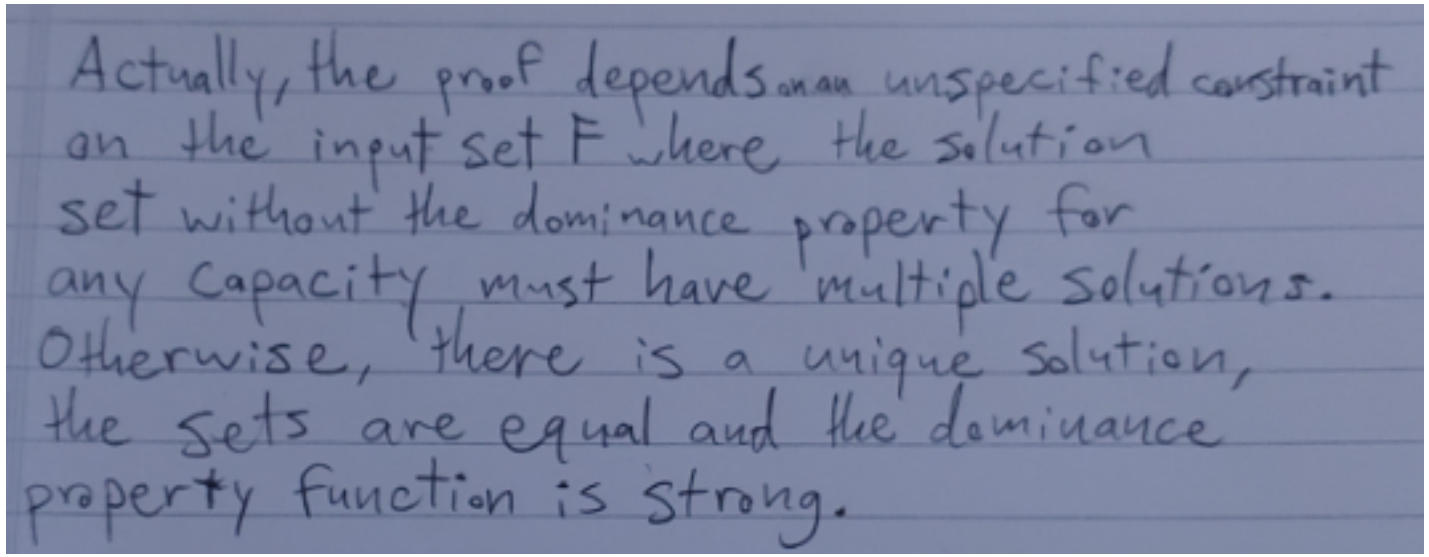




Subset (8)
Implication of the conjunction of relations {1, 2}.

Proper Subset (9)
Implication of the conjunction of relations {7, 8}.

Definition (10)
From relations {9}, Proof by definition



Proof(s) – Dominance Property Implies Extreme

The last thing that could be proven is that if (1) is true, then (2) is also true.

Proof(s) – Dominance Property Implies Extreme

$\forall I \in S', I \subseteq F$

$\forall I \in S', \sum_{j \in J \in I} f_j \leq D$

$\forall I \in S', |I'| \leq |F|$

$\forall J \in F - T_{k-1}, f_j \in J \geq f_i \in I, \forall I \in T_{k-1}$

$12 \wedge 13 \wedge 14 \rightarrow 2$

Subset (11)
Defining constraints on the solution set

Inequality (12)
Defining constraints on the solution set

Inequality (13)
From relations {11}, by definition of cardinality

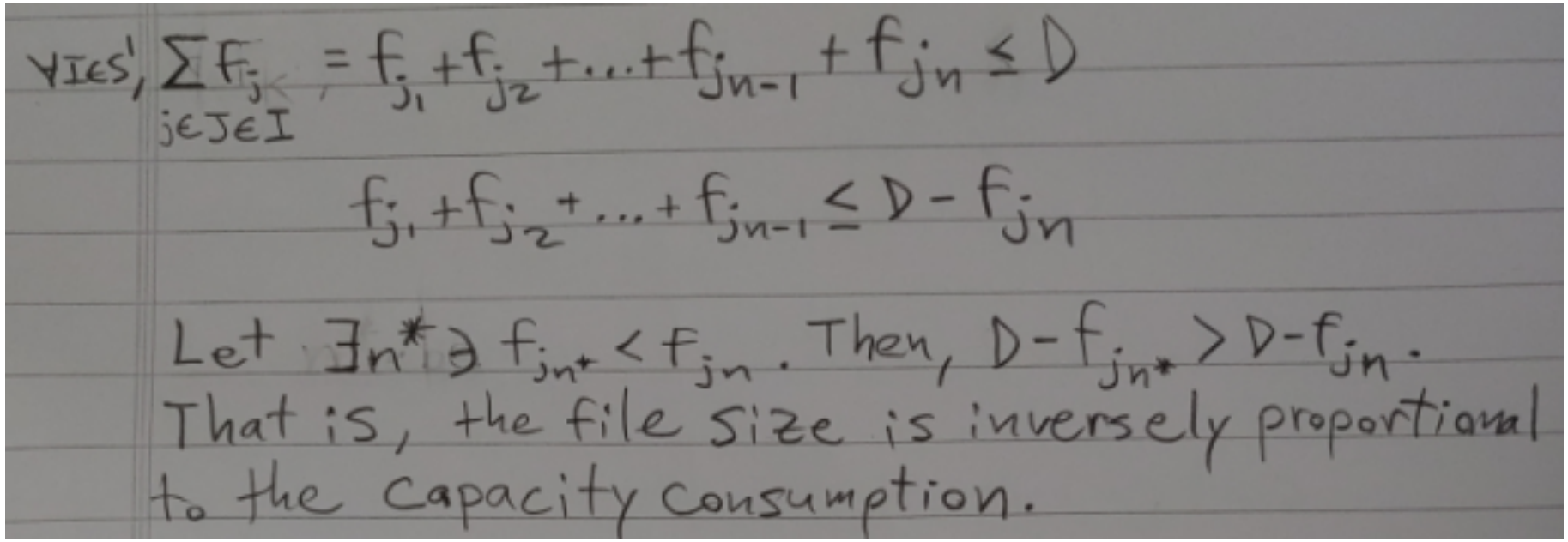
Inequality (14)
From relations {3, 4}, by Algebra and definition of minimum

Contradiction (15)
From relations {12, 13, 14}, by contradiction

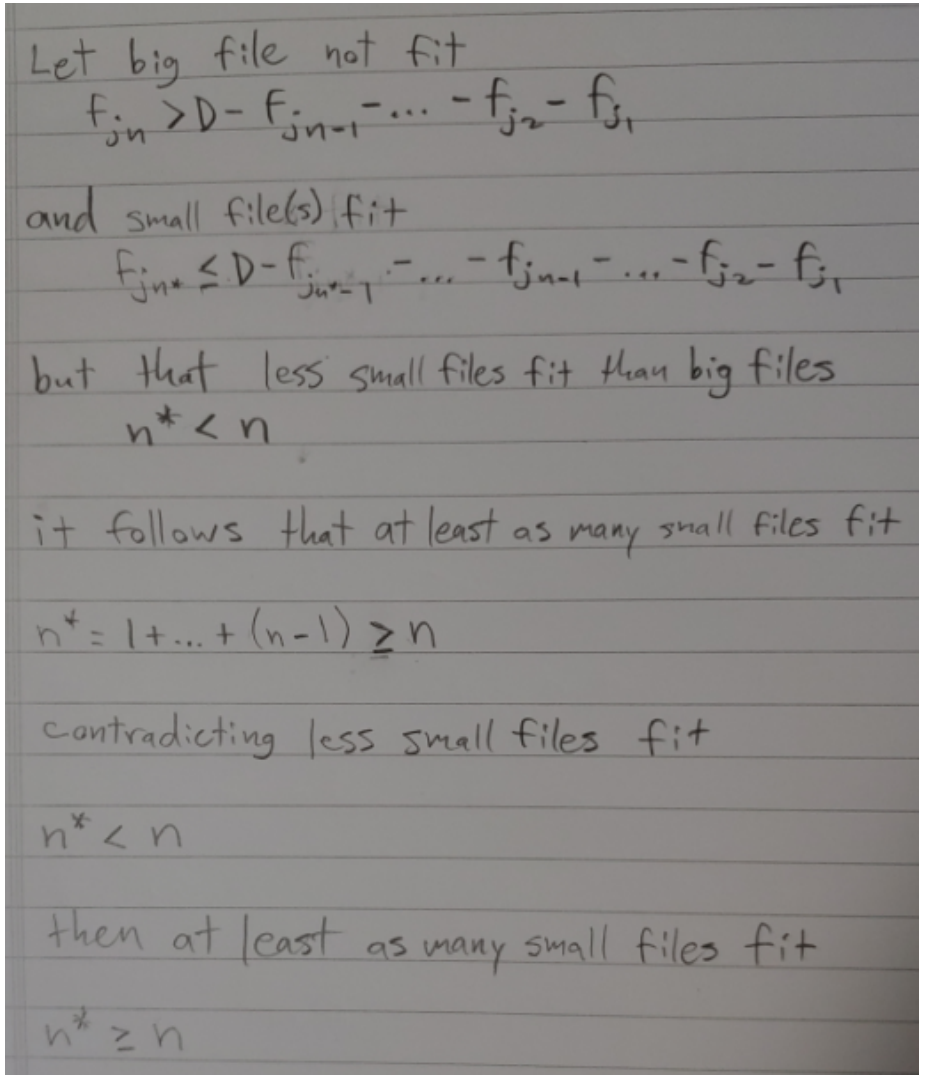
$12 \wedge 13 \wedge 14 \rightarrow \neg 2$

Choosing a larger file size, results in less capacity consumption so that more files can be consumed.

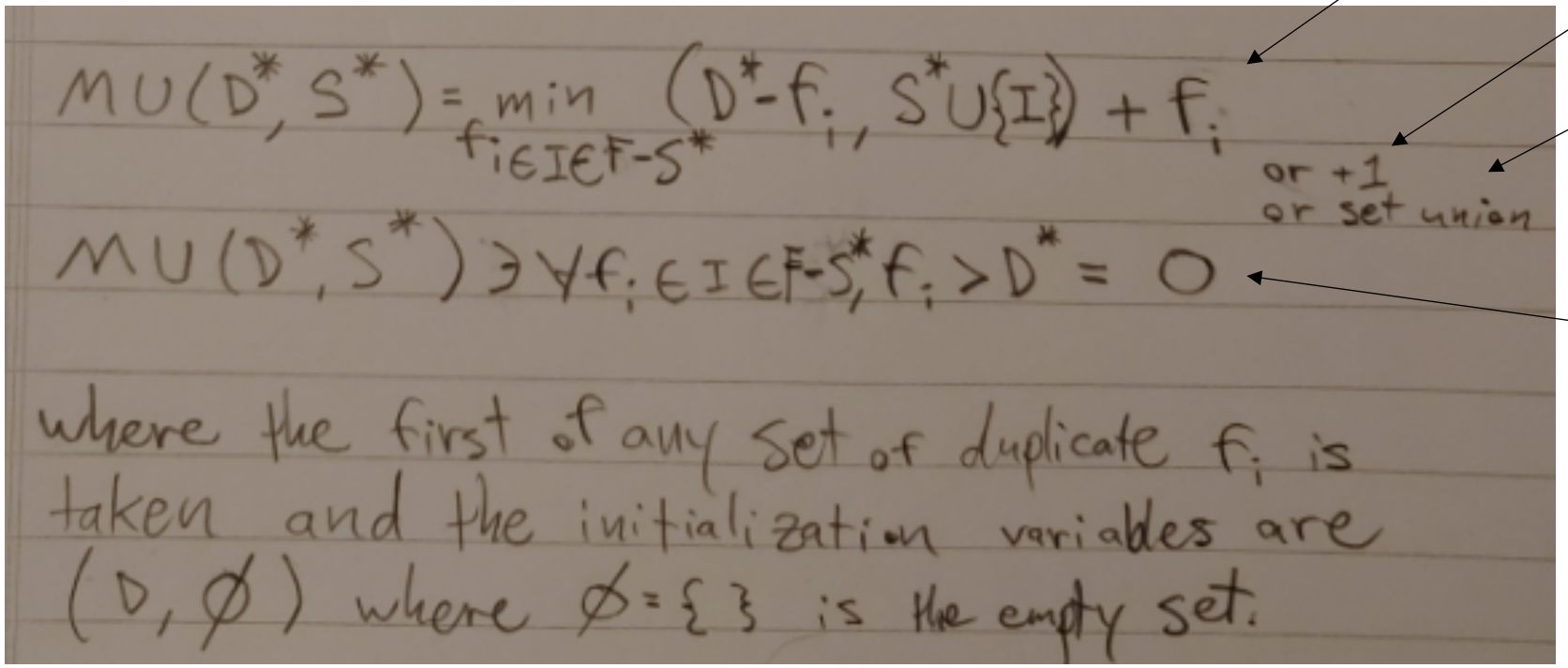
Proof(s) – Dominance Property Implies Extreme



Proof(s) – Dominance Property Implies Extreme



Formulas



Disk Space; File Size

Set Cardinality

Vector Set

or +1
or set union

Base Case

Complexity

Complexity without Dominance Property

$(O(n) \times O(n) \times O(1)) = O(n^2)$

Assume that the order of elements in F are sorted according to f_i, increasing.

Then, complexity with dominance property is:

$O(O(1) \times O(n) \times O(1)) = O(n)$

However, if the elements in F are randomly ordered with respect to f_i, then the complexity with dominance property is:

$O(O(1) \times O(n) \times O(n)) = O(n^2)$

Even a comparison sort with asymptotically optimal time complexity would be advantageous.

Sources

1: Antoine Jouglet, Jacques Carlier. Dominance rules in combinatorial optimization problems. European Journal of Operational Research, Elsevier, 2011, 212, pp.433-444. ff10.1016/j.ejor.2010.11.008ff. ffhal00625751

https://hal.archives-ouvertes.fr/hal-00625751/document

2: Advanced Algorithms: Greedy Algorithms, Spring 2020, Erik Saule