To Be Greedy or Not to Be?

Tal Mizrahi

Computer Science student

The Open university

Abstract

I will provide a characterization of the cases where the greedy algorithm may produce the best solution, the unique worst solution, and an approximated solution. Definitions such as independent systems, and matroids would be explained in this article to help demonstrate the phenomenal where the greedy algorithm will surely perfectly succeed.

1 Introduction

The Greedy algorithm is defined as a method for solving optimization problems where the goal is to make the locally optimal choice at each stage with the hope of finding a global optimum. It is called "greedy" because it tries to find the best solution by making the best choice at each step, without considering future steps or the consequences of the current decision. It works for cases where **minimization** or **maximization** leads to the required solution.

1.1 Characteristics of greedy algorithms:

- **Making locally optimal choices** – a greedy algorithm selects the best option available at that specific moment at each step without taking the decision's long-term implications into account.
- **No backtracking** – a greedy algorithm's decisions are final and cannot be changed or undone after they have been made. The algorithm keeps going without going back to its earlier choices.
- **Iterative process** – greedy algorithms operate in a succession of iterative phases, each building on the one before it.
- **Efficiency of greedy algorithms** – Greedy algorithms frequently have a low number of steps and are consequently computationally quick, they are often effective in terms of temporal complexity.

1.2 Some common use cases for the greedy algorithm include:

- **Scheduling and Resource Allocation:** The greedy algorithm can be used to schedule jobs or allocate resources in an efficient manner.
- **Minimum Spanning Trees (MST):** The greedy algorithm can be used to find the minimum spanning tree of a graph, which is the subgraph that connects all vertices with the minimum total edge weight.

- **Coin Change Problem:** The greedy algorithm can be used to make change for a given amount with the minimum number of coins, by always choosing the coin with the highest value that is less than the remaining amount to be changed.
- **Huffman Coding:** The greedy algorithm can be used to generate a prefix-free code for data compression, by constructing a binary tree in a way that the frequency of each character is taken into consideration.

While there's a scientific consensus that the greedy algorithm cannot solve all the optimization problems, there's a common belief that the greedy algorithm is worth implementing anyway because it provides a good approximation to the optimal solution quickly and efficiently. In this paper we will explore each occasion where the greedy algorithm will succeed finding the optimal solution, the unique worst solution, and an approximated solution.

2 Terminology and Notation

2.1 Independent system

An independence system is a pair consisting of a finite set E and a family F of subsets (called independent sets) of E such that the two followings are satisfied:

(1a) the empty set is in F.

(1b) **Heredity property:** If $X \in \mathsf{F}$ and $Y \subseteq X$, then $Y \in \mathsf{F}$.

All maximal sets of F are called bases.

2.2 Matroids

A Matroid is an independent system with one additional property.

(2a) **Exchange property –** if $A, B \in \mathsf{F}$ and $|A| < |B|$, then $\exists x \in B \backslash A$ such that $A \cup \{x\} \in \mathsf{F}$.

2.2.1 Minimum Spanning Trees (MST)

The MST problem is a specific example of an independent set withing the **graphic matroid** associated with a graph:

- **The Ground Set:** E is all the edges in the graph.
- **Independent Sets:** $\mathsf{F} \subseteq 2^E$, where each f ∈ F is a forest.
- **Empty Set:** is indeed a forest.
- **Heredity Property:** a sub-forest of a forest is also a forest.
- **Exchange Property:** For any A, B ∈ F, with B has more edges than A,then $\exists edge \in B \backslash A$ such that $A \cup \{edge\} \in \mathsf{F}$.

Proving the **exchange property** – assume negatively A is a forest but $A \cup \{e\}$ isn't a forest, then A would connect all vertices, thus it's a maximal set,

contradicting the fact the B is an independent set and has more edges than A. We already know that the greedy algorithm results in the optimal solution in the MST problem, let's just keep in mind that in its operation, the greedy would check if the forest is still a forest every iteration, caring not to violate the independent set property.

2.2.2 Job Scheduling Problem

- Consider a set of jobs, each with its own deadline and profit, also consider a machine that completes one job per time unit. The objective is to schedule the jobs in a way that maximizes total profit, with the constraint that each job must be completed by its deadline. The algorithm solving this problem is first sorting the jobs by profit in descending order, and then the greedy algorithm will place the highest profit job exists every local choice in its deadline as long as its deadline time slot is available. For example:
- J_1: Deadline = 2, profit = 20.
- J_2: Deadline = 2, profit = 15.
- J_3: Deadline = 1, profit = 10.
- J_4: Deadline = 3, profit = 5.
- J_5: Deadline = 3, profit = 1.

Timeslots:

Timeslot1: J_3	Timeslot2: J_1	Timeslot3: J_4

It's easy to see that this solution is indeed optimal for this specific occasion, but for proving that the greedy algorithm is optimal in this problem we'll have to present a more general claim about matroids and prove it. Let's first see why the Job Scheduling Problem is a Matroid.

- **The Ground Set:** E is all the Jobs available.
- **Independent Sets:** $\mathsf{F} \subseteq 2^E$, where each f $\in \mathsf{F}$ is subset of jobs that can be scheduled by their deadlines.
- **Empty Set is Independent Set:** The claim is vacuously true.
- **Heredity Property:** removing jobs doesn't create collisions of time units.
- **Exchange Property:** if an independent set A is larger than B, then B has at least one more timeslot available than A. Therefore, there must be at least one job in A that can be scheduled in the slots available for B.

Let's also keep in mind that the Greedy algorithm checks for the timeslot to be available each iteration, caring not to violate the independent set property.

This problem is strongly related to a real life important problem as the **job scheduling algorithm in cloud computing**, as a similar greedy-based algorithm proposed in [5] optimizing task allocation based on Quality of Service (QoS) parameters and fairness, which can be seen as analogous to maximizing profits while respecting deadlines in the general job scheduling problem.

3 Matroidal Structures

In the following theorem we'll characterize a general occasion of the greediness success in Matroid Maximization problems, so we'll be able to say that if a problem is a matroid, then the greedy algorithm will result the optimal solution. To do that, we'll show a general problem, and we'll do a reduction to it into a general claim for every problem who shares the same characteristics with it.

3.1 Matroid Maximization

As presented in [1] (pages 6-7), Let's consider a matroid $M = (E, \mathsf{F})$ and a weight function w: $E \to R$ (R for real numbers). For a subset A, w(A) = $\sum_{a \in A} w(a)$. The objective in finding a maximum weight independent set. The greedy algorithm will work as follows:

- **Initialization** – $S = \phi \in \mathsf{F}$
- **Iteratively adding elements –**
  - **For each** $e \in E$ taken in monotonically decreasing order do:
  - If $S \cup \{e\} \in \mathsf{F} \to S := S \cup \{e\}$.

Theorem 3.1 – for matroid maximization problems, the greedy algorithm is optimal.

Proof: Assume for contradiction, that there exists an optimal solution O with a greater weight than the solution S produced by the greedy algorithm $\exists O \in \mathsf{F}$ such that w(O)>w(S) and O is the optimal solution.
Let $s_1, s_2, \dots, s_k$ be S's elements and $o_1, o_2, \dots, o_m$ be O's elements.
Cardinality argument – let's show that $k = m$.
If $k > m$, then by the **exchange property**, there must be an element in S not in O. However, this contradicts the assumption that O is an optimal solution. Therefore, $k \not> m$.
If $m > k$, then by the **exchange property**, there must be an element in O not in group S. However, if such an element existed, then the greedy algorithm would have considered it, and adding it to S would violate the independence of S. Therefore, $m \not> k$.
Thus, S and O have the same cardinality, $m = k$.
$O = \{o_1, o_2, \dots, o_m\}$
$S = \{s_1, s_2, \dots s_m\}$
Assume that both O and S are sorted in the decreasing order of weights, $w(o_i) \geq w(o_{i+1})$ and $w(g_i) \geq w(g_{i+1})$. Since $S \neq O$ and O is assumed to be optimal, there must $\exists i$ where $w(o_i) > w(s_i)$. Find the smallest such i.
Up to the (i-1)th element, S and O are identical, i.e., $(s_1, s_2, \dots, s_{i-1}) = (o_1, o_2, \dots, o_{i-1})$.For the ith element, $s_i \neq o_i$,and $w(o_i) > w(s_i)$. However, this leads to a contradiction: the greedy algorithm would have chosen o_i over s_i at step i if o_i was a better choice(i.e., had a higher weight and maintained the independence of the set). This contradicts the assumption that S is the result of the greedy algorithm. Therefore, no such O can exist that has a

greater weight than S, proving that the greedy algorithm indeed produces the maximum weight independent set, the optimal solution.
Thus, if the problem fits to a matroid maximization problem → the greedy algorithm is optimal.■

3.1.1 Matroid as a proving tool

Based on **Theorem 3.1**, a way of proving that a greedy solution is optimal on any system, includes showing that the problem satisfies the properties of a matroid, and that each iteration of the greedy algorithm reserves the solution as an independent set.
For example, we've shown that the MST problem and the Job Scheduling problem both have the characteristics of a matroid, and for both the greedy algorithm reserved the properties of the independent set withing the solution. That's enough for proving that the greedy solution is the optimal solution for these problems.

3.2 Matroids – a Rich World

you might be surprised to hear that the number of distinct types of matroids is not fixed or easily quantifiable, as matroids can be defines and classified in numerous ways depending on the contest and the properties of interest. We'll mention some of them:
Graphic Matroids – These are associated with graphs. The independent sets are the forests in the given graph. such as the MST. another optimization problem solved with the greedy algorithm on a graphic matroid is the **Maximum Weight Forest**, where's the goal is to maximize the total weight of the selected edges while ensuring the subset of edges forms a forest.
Binary Matroids – These are representable over the binary field GF(2), z_2, graphic matroids are considered a special case of binary matroids.
Gammoids – These are Matroids that can be represented by directed graphs.
Transversal Matroids – can be associated as bipartite graph matching, such as the Job Scheduling Problem.
Algebraic Matroids – These are defined based on the algebraic dependencies over a field. An optimization problem for this field may be to find a base for a R^n vector space. The greedy algorithm will add vectors to the basis as long as their still linear independent.
Uniform Matroids – where a set is independent if and only if its size is below a certain threshold.
Lattice Matroids – these are related to lattice theory and involve a lattice of flats, which are certain types of substructures within the matroid.
Plenty more Matroids – Partition Matroids, Representable Matroids, Cographic Matroids, Regular Matroids, Isomorphic Matroids, Discrete Matroid and more…

As many as the types of Matroids are, the many problems can be solved with a greedy algorithm.

4 Non-Matroidal Structures

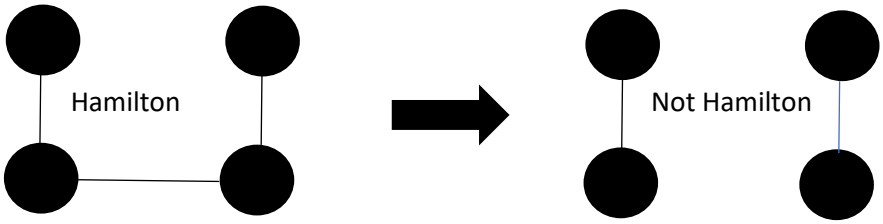
As we've seen, a matroid system ensures us an optimal solution for the greedy algorithm. In the followings, we'll examine non-Matroid structures such as the Symmetric Traveling Salesman Problem (we'll be mentioned as STSP) and the Minimum Coin Problem.

4.1 STSP

Given a complete graph(clique) with weights for each edge, Let the nodes represent cities and the weighted edged to represent distances between cities. The objective is to find the shortest possible route that visits each city exactly once, except the origin city, that should be visited twice, once at the beginning of the path and a second time of the ending of the path. Since we're talking about the symmetric problem, the distance from city A to city B is the same as the distance from city B to city A.

4.1.1 STSP – cannot be represented as an Independent System

Let's try to define independent sets in the STSP problem. These sets by definition must be a part of the solution, Hamilton cycles in our case, from the empty set to the whole solution. These sets won't be able to apply the **Heredity Property**. Take for example, the complete solution, by the **Heredity Property** we should be able to remove any element of it and still have an independent set in our hands. Yet, removing an element (in our case, edge), may construct a forest of two unconnected trees which can't be a part of the solution. There would also be a problem with the **Exchange Property**, but we'll skip the redundancy of showing that, since the system isn't independent it cannot be a Matroid.



4.1.2 The greediness failure

In the STSP problem, the greedy algorithm chooses the shortest route to the next city he hasn't visited yet. This approach might yield the unique **worst** optimal solution. Have a look at a set of nodes $\{S, A, B, C\}$ where's S is the starting node, and a set of weighted edges $\{S \to A = 1, A \to B = 2, B \to C = 10, S \to C = 2, S \to B = 2, A \to C = 1\}$. The greedy route would've be $S \to A \to C \to B \to S$ which will sum up to 1+1+10+2 = 14, yet the optimal route would've be $S \to$

$B \to A \to C \to S$ which will sum up to 2+2+1+2=7.In fact, the greedy route would've choose the same route even if the weight of edge $B \to C$ be 99^{99}. Thus, choosing the greedy route without special care might be dangerous.

4.1.3 – Special STSP case

There is a special condition presented in [2] where the greedy algorithm for certain won't produce the unique worst solution in the STSP. This condition is forcing a specific local junction where the greedy algorithm can't choose the worst possible solution.

Theorem 4.1.3 consider a restricted version of STSP optimization problem. If $n \geq 4$ and |W|$\leq \left\lfloor \frac{n-1}{2} \right\rfloor$, then the greedy algorithm never produces the unique worst possible solution.

Proof: Lemma 4.1.3 Since there's only $\left\lfloor \frac{n-1}{2} \right\rfloor$ different weights, there most exist two edges without any common nodes with the same weight 'k', call them x_i, x_j.

Base of induction – for $n = 4$ there's only 1 weight possible so there is no unique worst solution to be chosen.

Inductive step Assume the statement is true for a clique with n vertices, where $n \geq 4$. We need to show that it is also true for a clique with n+1 vertices.

in a clique of n+1 vertices, there are $\binom{n+1}{2} = \frac{n(n+1)}{2}$ edges. Since $|W| \leq \frac{n-1}{2}$ and we're considering n+1 vertices, the number of distinct weights is at most $\frac{n}{2}$. By the induction hypothesis, in the clique of n vertices, there are at least two edges with the same weight. Now, when we add the (n+1)th vertex, we add n new edges. The number of available distinct weights for these is at most $\frac{n}{2}$. For an even n, $\frac{n}{2}$ is an integer, and since we're adding n edges, by the Pigeonhole Principle, at least two of these edges must have the same weight.

The case of odd n is almost similar. This completes the inductive step, and the lemma.

Assume B = $\{x_1, x_2, \dots, x_n\}$ a solution of edges chosen by the greedy algorithm in sequence given by their indices. also assume that B is the unique worst solution. Assume x_i connects two nodes: $\{u, v\}$ and x_j connects two different nodes $\{x, y\}$.

We can assume that there's another solution B' that have the same edges as B with two changes: adding the edges (u, y) and (x, v) and restricting the edges $(u, v), (x, y)$.

Both new edges must have a weight greater than 'k' since otherwise, the greedy algorithm would have locally chosen one of these edges instead of x_i or x_j.

Thus, $w(B') \geq w(B)$, and the greedy algorithm will never produce the unique worst possible solution under these restricted conditions.■

4.2 – Minimum Coin Change Problem – special sets

The Conclusion so far sounds simple, right? If a system can be defined as a matroid, the greedy algorithm is optimal, else, it is not. Well, as will be proved now, things aren't so simple.

As presented in [3], the Change-Making Problem goal is to represent a given value with the fewest coins under a given coin system. It is known to be NP-hard. Nevertheless, in most real money systems, the greedy algorithm, which chooses the largest coin each step, yields optimal solutions.

Definition: A coin system $ is canonical if $|GRD_{\$}(x)| = |OPT_{\$}(x)|$.

Multiplication Factor - in this coin system, each denomination is a multiple of the previous. Because of the multiplicative relationship. Any combination of lower denomination coins that adds up to a higher denomination can be replaced by a single coin of that higher denomination.

Claim: the coin system $\$ = \{1,5,10,25\}$, which applies the **Multiplication Factor** is canonic.

Proof: Assume an alternative optimal solution that uses a different combination of coins than the greedy solution and with a fewer number of coins. If the alternative solution uses fewer 25-cent coins, it must make up the difference with a combination of 10-cent,5-cent coins. However, any combination of these coins that totals 25 cents or more would require more coins than a single 25-cent coin. Contradicting the assumption of fewer coins. Similarly, using fewer 10 cent coins would necessitate more 5-cent coins, and the argument follows as above thanks to the multiplication factor. Therefore, the greedy algorithm must be optimal for the coin system {1,5,10,25}, as it always uses the least number of coins to make up any given amount, and the coin system is canonic.■

Some might infer that the Minimum Coin Change Problem can be represented as a matroid since the greedy algorithm yields an optimal solution. Yet, that's not the case. If the system could've represented a matroid, then the greedy solution would've yielded the optimal solution for every possible ground set and weight function, without considering the **Multiplication Factor**.

For example, look at this set of coins – $\{1,3,4\}$. Since 4 isn't a multiplication of 3, it doesn't apply the **Multiplication Factor**. To sum $x = 6$, the greedy approach will use the set $|\{4,1,1\}| = 3$ coins, while the optimal solution is $|\{3,3\}| = 2$ coins. Thus, the set of coins is non-canonical, the greedy algorithm doesn't provide an optimal solution, hence from **Theorem 3.1**, the Minimum Coin Change Problem cannot be represented as a Matroid.

From this section we can learn that using specific properties on some problems such as the Multiplication Factor in the Minimum Coin Change Problem may cause the greedy solution to be the optimal solution, even if the

system cannot be represented as a Matroid. In conclusion, if the greedy algorithm is optimal ⊣ The system is a Matroid.

5. k-extendible systems

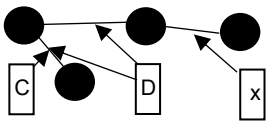
In this section we study greedy algorithm as an approximation algorithm, as presented in [4].

Definition. The subset independent system (E, F) is **k-extendible** if for all $C \in$ F and $x \notin C$ such that $C + x \in$ F and for every extension D of C ($C \subseteq D$) there exists a subset $Y \subseteq D - C$ with $|Y| \leq$ k such that $D - Y + x \in$ F.

Theorem 5. The system (E, F) is a matroid is and only if it's 1-extendible.

Proof. Part 1: Matroid → 1-Extendible. Take sets $C \subset D$ in F and an element $x \notin D$. We need to find a set Y such that $D - Y + x$ is in F.

Case 1: if $|C + x| = |D|$, then C and D differ by one element. Set $Y = D - C$,which has only one element. Then $D \backslash Y + x = C + x$, which is independent.



Take the MST as a Helping illustration for case 1.

Case 2: if $|C + x| \neq |D|$, use the **Exchange Property** of the Matroid to add elements from $D - C$ to $C + x$ until we get case 1.

Part 2: 1-Extendible→Matroid. Given two independent sets A,$B \in$ F such that $|A| < |B|$ we need to find $z \in B - A$ that $A + z \in$ F.

Case 1: if $A \subseteq B$, any $z \in B - A$ is fine, using the **Heredity Property** of independent systems.

Case 2: if $A \nsubseteq B$, pick $x \in A - B$. If $B + x$ is independent, we're done. Otherwise, set $C = A \cap B$ and $D = B$. Since the system is 1-extendible, there exists a single element $y \in B - A$ ($y \in D - C$) such that $B - y + x$ ($D - y + x$) is an independent set. Replace y with x in B and repeat this process until $A \subseteq B$.■

5.1 The $\frac{1}{k}$ factor approximation

Definition. a $\frac{1}{k}$ **factor approximation** refers to a specific performance guarantee. For example, when an algorithm is said to produce $\frac{1}{2}$-approximate solution for a problem, it means that the value obtained by the algorithm is at least 50% as effective as the optimal solution for the problem. This is expressed as: $ALG \geq \frac{1}{2} \cdot OPT$.

Theorem 5.1 – if (E,F) is k-extendible, the greedy algorithm is $\frac{1}{k}$ factor approximation for the optimization problem defined by (E, F) and any weight function w.

Proof. The greedy algorithm picks elements $x_1, x_2, \dots, x_l$ in a sequence. For each step i, the solution S_i is formed by adding the element x_i to the previous Solution S_{i-1},starting from an empty set $S_0 = \phi$. To continuate the proof we need the following lemma whose proof we defer because it's exceeding this paper subject.

Lemma 5.1 if (E,F) is k-extendible, then the ith element x_i picked by the greedy algorithm is such that $w\big(OPT(S_{i-1})\big) \leq w\big(OPT(S_i)\big) + (k-1) \cdot w(x_i)$.

The optimal solution starting from an empty set referred as $OPT(\phi)$, which in our case is $OPT(S_0)$. By the applying lemma 5.1 l times we get: $w\big(OPT(S_0)\big) \leq w\big(OPT(S_1)\big) + (k-1)w(x_1) \leq w\big(OPT(S_2)\big) + (k-1)\big(w(x_1) + w(x_2)\big) \leq \cdots \leq w\big(OPT(S_l)\big) + (k-1)\sum_{i=1}^{l} w(x_i)$.

We can replace $W\big(OPT(S_l)\big)$ with $w(S_l)$ because the set S_l is maximal. Then we get $w\big(OPT(S_0)\big) \leq w(S_l) + (k-1) \cdot w(S_l) = kw(S_l)$, and divided by k, $\frac{1}{k} w\big(OPT(S_0)\big) \leq w(S_l)$.

Hence the greedy solution is a good approximation of the optimal solution. The Theorem is proved. ■

Specifically, the conclusion the of proof indicates that the greedy algorithm provides a solution that is at worst k times the weight of the optimal solution, or conversely, at least $\frac{1}{k}$ times the weight of the optimal solution. As we know from Theorem 3.1, the greedy algorithm is optimal for matroid maximization problems, and concluding Theorem 5 with Theorem 5.1, we get $w(S_l) \geq w\big(OPT(S_0)\big)$,a result which aligns with our paper.

5.2 Matroid Intersection

Theorem 5.2 The intersection of k matroids is k-extendible.

Proof. Let a set E and k Matroids. Each matroid is represented as (E, f_i) for $1 \leq i \leq k$. The intersection of all these matroids, $\cap_i fi$, is denoted as F. We need to show that for every independent set $C \subseteq D \in$ F and $x \notin C$ such that $C + x \in$ F there exist $Y \subseteq D - C$ with at most k elements such that $D - Y + x \in$ F. Since the above sets are in F, they are also in f_i. By Theorem 5 these individual matroids are 1-extendible, this means for each matroid, we can find a small set (at most one element), Y_i, such that, when removed and replaced with x, keeps the set in the matroid($D - Y_i + x \in f_i$). Set $Y = \cup_i Y_i$, and since Y_i has at most one element and there are k matroids, Y will have at most k elements, and for all i we have $D - Y + x \in f_i$, which implies independence with respect to F.■

This last theorem shows a nice relationship between matroids and k-extendible systems.

5.2.2 Maximum Weight Bipartite Matching

The Maximum Weight Bipartite Matching is an example of a 2-extendible problem of two matroids (this won't be proven here). In a bipartite graph, the vertices are divided into two disjoint sets, and each edge connects a vertex from one set to a vertex in the other set. The goal of the Maximum Weight Bipartite Matching problem is to find a matching, a set of edged in which no two edges share a common vertex, that has the maximum possible sum of weights, where each edge is assigned a certain weight. Let the ground set E of both matroids be the edges in the bipartite graph.

Matroid 1: this matroid is based on the vertices on one side of the bipartite graph. An independent set in this matroid is a set of edges where no two edges share the same vertex on this side of the graph. **Matroid 2** is declared similarly for the other side.

The greedy algorithm always picks the heaviest edge that can be added to the current matching M without violating the matching constraints. We know this solution isn't optimal, but let's show why it's a $\frac{1}{2}$-factor approximation.

Edge comparison – for each edge e^* which is not in the solution M but in an optimal matching M^*, there must be at least one edge, e, in M that shares a vertex with e^*, because if no edge existed, the greedy algorithm would have included e^* in M. By the nature of the greedy algorithm, $w(e) \geq w(e^*)$.

Bounding the Optimal Solution – Consider the sum of the weights of all edges in M^*. Since each edge $e^* \in M^*$ that is not in M is associated with at least one edge $e \in M$ with $w(e) \geq w(e^*)$, we can say that the total weight of M is at least half of the total weight of M^*. Formally, $\sum_{e \in M} w(e) \geq \frac{1}{2} \sum_{e^* \in M^*} w(e^*)$.

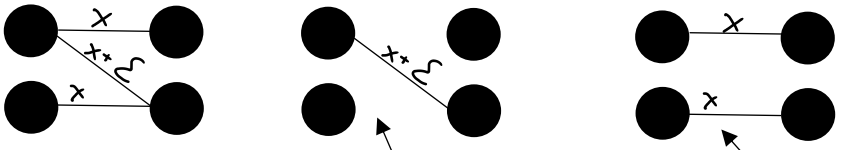


Illustration – a bipartite-graph maximization, $\varepsilon \to 0$.

The Greedy matching is at least $\frac{1}{2}$ as good as the Optimal.

It's crucial to understand that for every edge in the optimal solution that the greedy algorithm misses, there's a corresponding edge in the greedy matching that "blocks" it and has a weight that is at least as large. This insight guarantees that the total weight of the greedy matching is not less than half of the total weight of the optimal solution. The Maximum Weight Bipartite Match problem has numerous applications, such as in job assignment, network flows, and the study of marriages and stable allocations.

Conclusion

We introduced the notion of the greedy algorithm on non-independent systems, Independent Systems, Matroids, and Matroids-intersections. We

can now say for certain that for Matroid Maximization problems, the greedy algorithm will provide the best solution. For matroid intersections we have a limit for how fat the greedy solution would be from the optimal solution. For more complex problems, it is still investigated whether or whether not we should use the greedy algorithm. Therefore, we should take caution and consider the trade-offs of choosing a good complexity algorithm, such as the greedy algorithm verses a more complex algorithm which will provide the optimal solution.

References

[1] Ulrich Faigle, Kyoto University, “A General Model for Matroids and the Greedy Algorithm”, Japan, 2007.

[2] Jorgen Bang-Jensen, Gregory Gutin, Anders Yeo, “When the greedy algorithm fails”, Odense, Denmark, 2004.

[3] Xuan Cai, Shanghai Jiao Tong University, “Canonical Coin Systems for Change-Making Problems”, Shanghai, China.

[4] Julian Mestre, “Greedy in Approximation Algorithms”, University of Maryland, College Park, MD 20742.

[5] Ji Li, Longhua Feng, Shenglong Fang, Chongquing University, “An Greedy-Based Job Scheduling Algorithm in Cloud Computing”, China, 2014.