Increasing profit from fees of cryptocurrencies

Besart Dollma   Noa Oved
Advisor: Itay Tsabary

Technion - Israel Institute of Technology

October, 2018

Outline

Outline

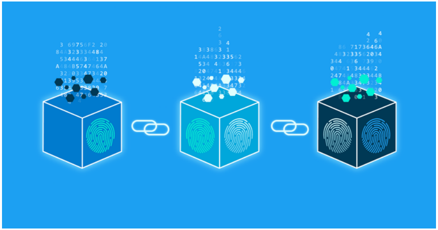
Cryptocurrencies

- A cryptocurrency is digital asset designed to work as a medium of exchange that uses strong cryptography to secure financial transactions, control the creation of additional units, and verify the transfer of assets.

- We will focus on Bitcoin.

Blockchain

- The validity of each cryptocurrency's coins is provided by a blockchain.
- Each block typically contains a hash pointer as a link to a previous block, a timestamp and transaction data.



Blockchain

- Resistant to data modification.
- On average, every 10 minutes a new block is added to the blockchain.
- Block size is 1 MB.

Transactions

- When transferring cryptocurrency from one party to another, a transaction is created and added to the Mempool (explained later).
- Among other things, transactions contain:

ID	672e2c74d410d0a5b689925155098c9a39
Fee	0.00015820 BTC
Size	224 bytes
Depends	[ ]

MemPool

- Mempool (a compound of two words, ‘Memory’ and ‘Pool’) is a pool of memorized, held data.
- The data that is being stored on the Mempool are unconfirmed transactions that are currently stuck on the network.
- We will denote the size of the mempool by n.
- $n \approx 16000$.

Miners

- In cryptocurrency networks, mining is a validation of transactions.
- For this effort, successful miners obtain new cryptocurrency as a reward.
- For each transcation the miner includes in a block, he collects its fee.
- Hence, the miner's motivation is to maximize the sum of the fees of the transactions that he includes in the block. This needs to be done under the size and dependency constraints. The miner strives to select the optimal set of transactions.

Outline

The knapsack problem

Definition

We are given a knapsack (block) of size W and n transactions $\{a_1, a_2, ..., a_n\}$.
Each transaction a_i has size $s_i > 0$ and fee $f_i > 0$.
We are to find $I \subseteq [n]$ such that:

$$I = \arg\max_{J \subseteq [n]} \{\sum_{j \in J} f_j\}$$

such that

$$\sum_{j \in J} s_j \leq W$$

The dependency knapsack problem

- The input is a set of transactions, each with a size and fee, a set of dependencies between the transactions, where no circular dependencies exist and a total knapsack (block) capacity of W.
- We will treat the input as a directed acyclic graph $G = (V, E)$.
- Each node v represents a transaction that has fee $f(v)$ and size $s(v)$.
- Each edge (i, j) represents that transaction j is dependent upon transaction i.
- Transaction j can be selected if transaction m is selected $\forall (m, j) \in E$.

The dependency knapsack problem

Definition

The goal is to find $\bar{V} \subseteq V$ such that:

$$\bar{V} = \arg\max_{J \subseteq V} \{ \sum_{v \in J} f(v) \}$$

such that

$$\sum_{v \in J} s(v) \leq W$$

and the dependency constraints are preserved $\forall v \in \bar{V}$.

The dependency knapsack problem

Example

- The knapsack size is $W = 11$.

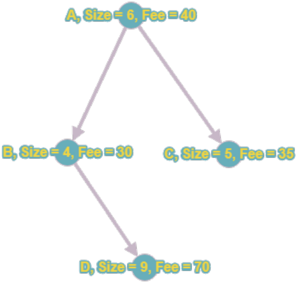


Figure: Example of Knapsack with dependencies

- Every transaction is dependent upon A, hence A must be selected.

The dependency knapsack problem

Example

- The knapsack size is $W = 11$.

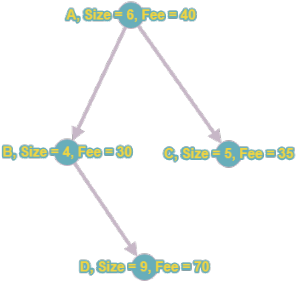


Figure: Example of Knapsack with dependencies

- Every transaction is dependent upon A, hence A must be selected. Optimal solution is $\{A, C\}$.

NP-Completeness

- The decision problem form of the knapsack problem is NP-complete.
- The dependency knapsack problem is a generalization of the knapsack problem.
- As such, the decision problem form of the dependency knapsack problem is also NP-Complete.
- Thus there is no known algorithm both correct and polynomial for any of the problems.

Outline

Knapsack solvers

Exhaustive Search

- Trivial approach.
- Check all the subsets $J \subseteq [n]$.
- Optimal solution.
- Runtime is $\mathcal{O}(2^n)$, hence exponential.
- We remind that $n \approx 16000$, therefore infeasible.

Knapsack solvers
Dynamic Programming

- Assume $s_1, s_2, \ldots, s_n, W \in \mathbb{N}$. It holds on real data.
- For each $k \in \{0, 1, \ldots, n\}$ and for each $w \in \{0, 1, \ldots, W\}$ define $F(k, w)$ to be the maximal profit when choosing from transactions $\{a_1, \ldots, a_k\}$ and the size of the block is w.
- It holds:

$$F(k, w) = \begin{cases} 0 & k = 0 \text{ or } w = 0 \\ F(k-1, w) & w, k > 0 \text{ and } s_k > w \\ \xi & w, k > 0 \text{ and } s_k \leq w \end{cases}$$

$$\xi = \max\{F(k-1, w), f_k + F(k-1, w - s_k)\}$$

- Optimal solution.
- Runtime is $\mathcal{O}(nW)$, hence pseudo-polynomial.

Knapsack solvers

Greedy approximation

- Sort the transactions by the $\frac{f_i}{s_i}$ ratio in descending order.
- Iterate in this order and add transactions to the block until the next transaction can't be added.
- Not optimal but a 2-approx.
- Runtime is $\mathcal{O}(n \log n)$.

Knapsack solvers

$(1+\varepsilon)$ approximation

- Let $0 \leq \varepsilon \leq 1$.
- Denote by *greedySol* the value of the greedy approximation and let

$$a = \varepsilon \cdot greedySol$$

- Denote

$$V_a = \{a_i \mid f_i < a\}$$

$$V_a^C = \{a_i \mid f_i \geq a\}$$

- For each $J \subseteq V_a^C$ such that $|J| \leq \frac{2}{\varepsilon}$:
  - Run the greedy approximation on V_a with block size $W' = W - \sum_{j \in J} s_j$ and denote the solution by I_j.
- Output $I_j \cup J$ with maximal profit.

Knapsack solvers

$(1 + \varepsilon)$ approximation

- Not optimal but a $(1 + \varepsilon)$ approx.
- Runtime is $\mathcal{O}(n^{1+\frac{2}{\varepsilon}} \cdot \log n)$.
- If $\varepsilon \to 0$ we receive the exhaustive search. Indeed it holds that $\frac{2}{\varepsilon} \to \infty$ (non-polynomial).
- If $\varepsilon \to 1$ we receive the greedy approximation. However the runtime is longer.

Solutions for the dependency knapsack problem

- We will discuss only the two approximations.
- The idea behind the algorithms remains the same.
- The algorithms are adjusted to supply the dependency constraints.

Definition

For each transaction v (node in the graph G)

$$Ancestor(v) \triangleq \{j \mid \text{there exists a path from } j \text{ to } v \text{ in } G\}$$

Note

Pay attention that $v \in Ancestor(v)$.

Dependency knapsack solvers

Greedy approximation

Algorithm

1. Calculate the sets $Ancestor(v)$ for all $v \in V$.
2. Pick $Ancestor(v)$ with the maximal

$$\frac{\sum_{j \in Ancestor(v)} f(j)}{\sum_{j \in Ancestor(v)} s(j)}$$

   ratio and add the transactions of the set to the knapsack.
3. Remove the transactions we just added to the knapsack from other sets and continue from 2 until we can't fit anything in the block.

- Runtime is $\mathcal{O}(n^3)$.

Dependency knapsack solvers

Greedy approximation

Example

- The knapsack size is $W = 11$.

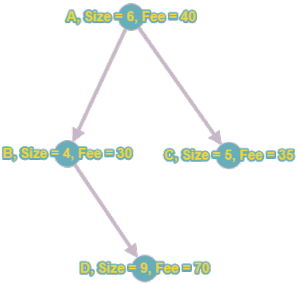


Figure: Example of Knapsack with dependencies

- We remind that the optimal solution is $\{A, C\}$.

Dependency knapsack solvers

Greedy approximation

Example

- $Ancestor(D) = \{D, B, A\}$ and $Ratio(D) = \frac{140}{19} \approx 7.36$.
- $Ancestor(B) = \{B, A\}$ and $Ratio(B) = \frac{70}{10} = 7$.
- $Ancestor(C) = \{C, A\}$ and $Ratio(C) = \frac{75}{11} \approx 6.81$.
- $Ancestor(A) = \{A\}$ and $Ratio(A) = \frac{40}{6} \approx 6.66$.
- The greedy approximation will try to add $Ancestor(D)$ to the knapsack but won't succeed since the size of the set is 19 and the knapsack size is 11.
- The algorithm will then add $Ancestor(B)$ to the solution and remove $\{B, A\}$ from the other sets.
- The algorithm will output $\{A, B\}$. Output value is 70.
- Not optimal.

Dependency knapsack solvers

$(1 + \varepsilon)$ approximation

- Similar to the $(1 + \varepsilon)$ approximation for the knapsack problem.
- Adjusted to the dependency knapsack problem using $Ancestor(\cdot)$ sets.
- Runtime is $\mathcal{O}(n^{3+\frac{2}{\varepsilon}})$.

Dependency knapsack solvers

$(1+\varepsilon)$ approximation

Example

- Let $\varepsilon = 0.1$
- $a = \varepsilon \cdot greedySol = 7$
- $$V_a^C = \{a_i \mid f_i \geq a\} = \{A, B, C, D\}$$
- Therefore we will check all $J \subseteq V_a^C$ such that $|J| \leq \frac{2}{\varepsilon} = 20$, in particular $\{A, C\}$.
- In this example the algorithm outputs the optimal solution.

An incremental solution to the greedy approximation

- Suppose at time t we solved the dependency knapsack problem using the greedy approximation algorithm.
- Now suppose at time $t + 1$ we added some transactions to the mempool and didn't remove any.
- Suppose also that the added transactions may be dependent on transactions from time t but not vice verse.
- Can we solve the dependency knapsack at time $t + 1$ using the solution at time t and reduce the runtime?

An incremental solution to the greedy approximation

- Since the transactions at time t can not be dependent upon transactions that are added at time $t+1$, $Ancestor(v)$ remains identical for all the transactions at time t.
- The idea of the incremental solution is to improve the running time.
- The solution value is the same as in the greedy approximation.

An incremental solution to the greedy approximation

Algorithm

1. Calculate $Ancestor(v)$ for all transactions v that weren't in the previous solution.
2. Denote by

$$\alpha = \max_{v} \frac{\sum_{j \in Ancestor(v)} f(j)}{\sum_{j \in Ancestor(v)} s(j)}$$

   the maximal ratio of fee over size of the transactions that weren't in the previous solution.
3. Add to the solution all the $Ancestor(\cdot)$ sets with a ratio bigger than α. Use the greedy approximation algorithm on what is left on the mempool with the new size of the block.

Outline

Implementation

- We implemented the algorithms using Python.
- Graphs aren't build in, hence we used the networkx package.
- Everything can be found in our github.
- Caching was disabled through out the tests.

Outline

1. Cryptocurrencies
2. The knapsack and dependency knapsack problems
3. Solutions
4. Implementation
5. **Results**
6. Conclusions

The knapsack problem
Experiment I

- Let's compare the knapsack solvers head to head.

Experiment I

For each $1 \leq n \leq 20$, we generated each time n random transactions a_i such that $1 \leq s_i, f_i \leq 100$. $W = 1000$. The data points are calculated as the average of 50 runs.

The knapsack problem
Experiment I

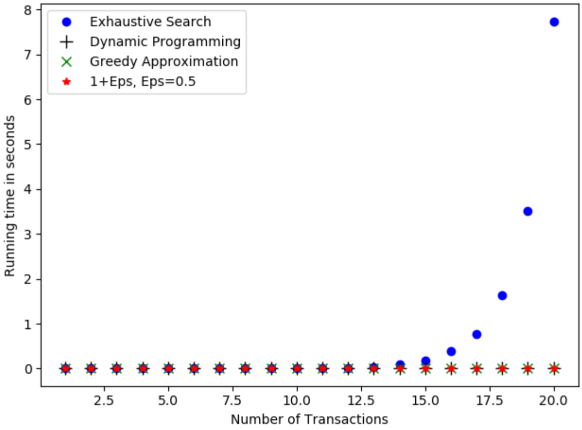


Figure: Runtime in seconds as a function of n

The knapsack problem

Experiment I

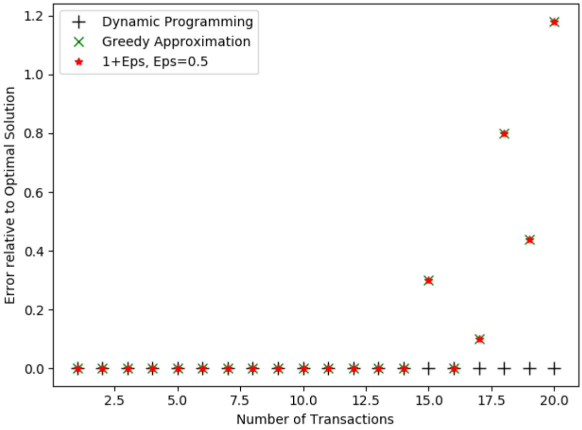


Figure: Relative error as a function of n

The knapsack problem

Experiment II

- Experiment I shows that exhaustive search is not feasible.
- Let's see how the other algorithms behave when the number of transactions gets bigger.

Experiment II

The parameters are the same as in Experiment I, only this time $n = 1, 26, 51, ..., 976$.

The knapsack problem

Experiment II

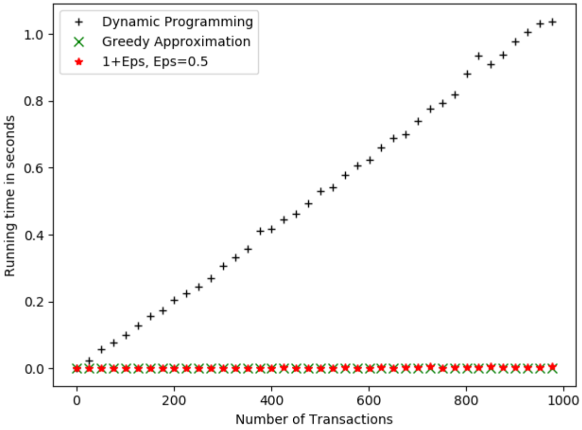


Figure: Runtime in seconds as a function of n

The knapsack problem

Experiment II

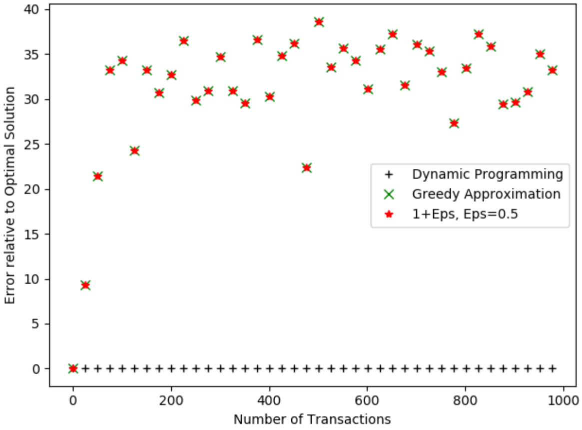


Figure: Relative error as a function of n

The knapsack problem
Experiment III

- From experiment II it looks like the dynamic programming might handle a case of real data.
- It is optimal and it took around one second to solve an instance of 1000 transactions.
- However let's alter the parameters a bit.

Experiment III

$$W = 10^6 = 1 \text{ Mega}$$

For each $1 \leq n \leq 25$, we generated each time n random transactions a_i such that $1 \leq s_i \leq 500$ and $1 \leq f_i \leq 2000$. The data points are calculated as the average of 25 runs.

The knapsack problem

Experiment III

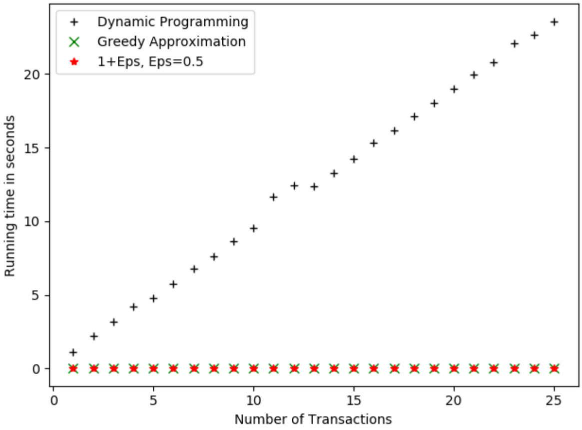


Figure: Runtime in seconds as a function of n

The knapsack problem

Experiment IV

- Experiment III shows that dynamic programming is also infesiable.
- We are left with the two approximations.

Experiment IV

For each $n = 1, 21, 41, ..., 981$ we generated each time n random transactions a_i such that $1 \leq s_i \leq 200$ and $1 \leq f_i \leq 100$. $W = 1000$. The data points are calculated as the average of 50 runs.

The knapsack problem

Experiment IV

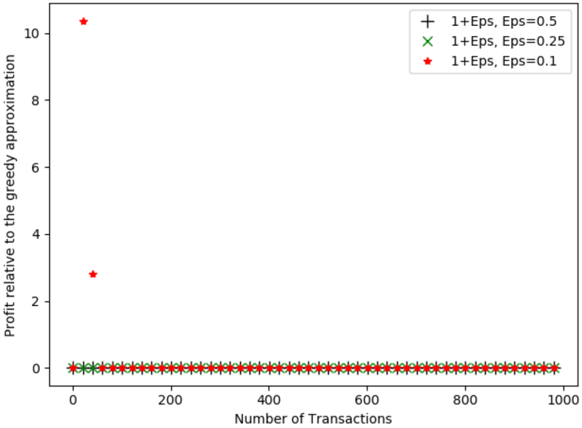


Figure: Profit compared to greedy approximation as a function of n

The knapsack problem
Experiment IV

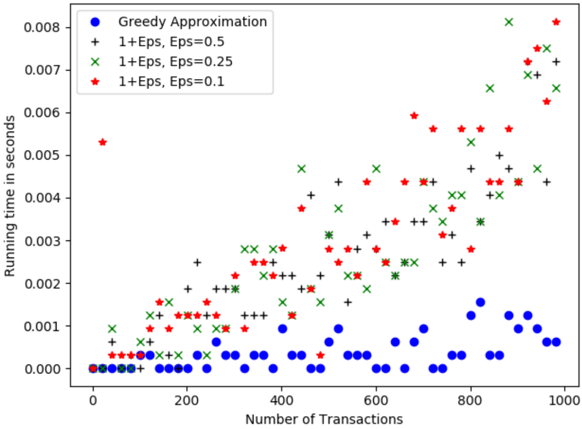


Figure: Runtime in seconds as a function of n

The knapsack problem

- Experiment IV shows that picking the right ε might increase the profit.
- However the right ε might be very small. This means that $\frac{2}{\varepsilon}$ might be very big.
- We remind that the runtime of the $(1+\varepsilon)$ approximation is exponential on $\frac{2}{\varepsilon}$.
- Hence $|V_a^C|$ shouldn't be too big, otherwise the algorithm won't be feasible.
- If $|V_a^C| > 20$ we reduce it to 20 by moving the other transactions to V_a. Now the algorithm isn't a $(1+\varepsilon)$ approx.
- We studied the following reduce parameters:
    - Pick 20 random transactions.
    - Pick the 20 transactions with the best $\frac{f_i}{s_i}$ ratio.
    - Pick the 20 transactions with the highest fee.
    - Pick the 20 transactions with the biggest size.

The knapsack problem
Experiment V

Experiment V

For each $n = 1, 11, 21, \ldots, 91$ we generated each time n random transactions a_i such that $1 \leq s_i \leq 200$ and $1 \leq f_i \leq 100$. $W = 1000$. The data points are calculated as the average of 5 runs.

The knapsack problem

Experiment V

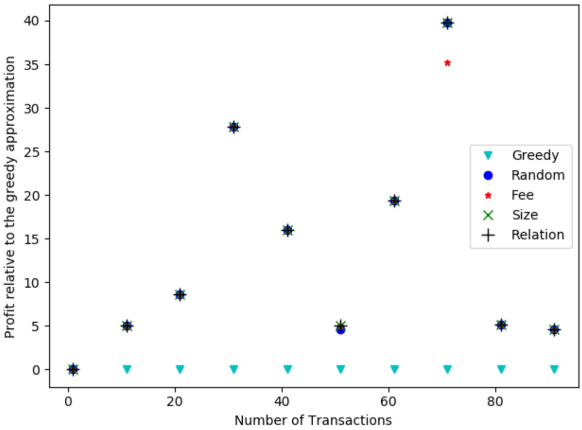


Figure: Profit relative to the greedy approximation as a function of n

The knapsack problem

Experiment VI

- How does ε influence the runtime and solution value of the $(1+\varepsilon)$ approximation?

Experiment VI

We randomly generated 50 transactions such that $1 \leq s_i, f_i \leq 200$. $W = 1000$. We decrease ε using

$$\varepsilon = \varepsilon \cdot 0.8$$

until $\varepsilon \geq 0.01$.

The knapsack problem

Experiment VI

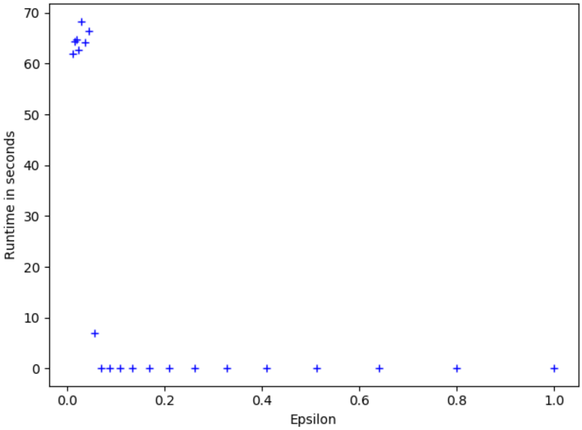


Figure: Runtime in seconds as a function of ε

The knapsack problem

Experiment VI

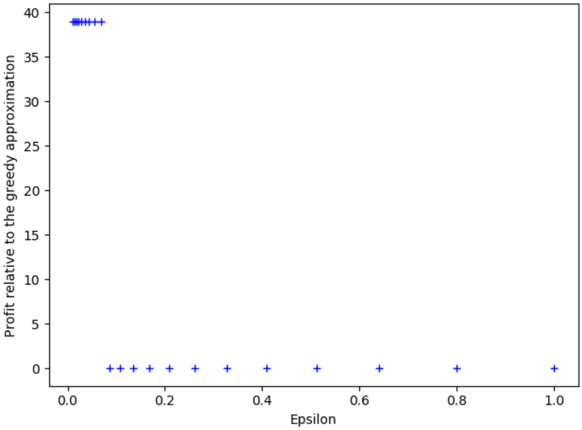


Figure: Profit compared to greedy as a function of ε

The dependency knapsack problem

Experiment I

- How do dependencies influence the runtime of the algorithms?
- The greedy approximation for the knapsack runs $\mathcal{O}(n \cdot \log n)$.
- The greedy approximation for the dependency knapsack runs $\mathcal{O}(n^3)$.

Experiment I

For each $n = 1, 11, 21, ..., 141$ we generated each time n random transactions a_i such that $1 \leq s_i, f_i \leq 100$. $W = 10000$. The dependencies are also generated randomly such that there are no circular dependencies. The data points are calculated as the average of 10 runs.

The dependency knapsack problem
Experiment I

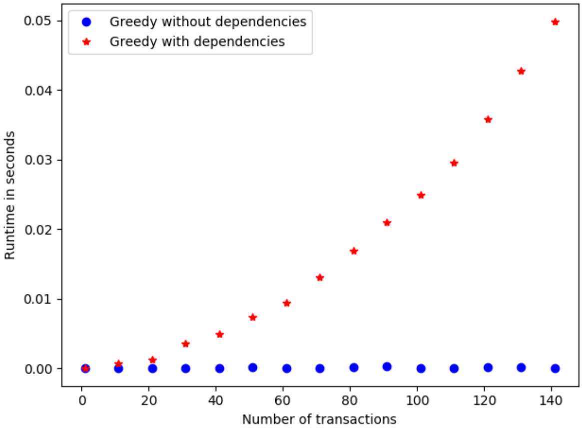


Figure: Runtime in seconds as a function of n

The dependency knapsack problem

Experiment II

- We want to compare between the two approximations.
- Also we want to explore the $(1+\varepsilon)$ approximation for different ε.

Experiment II

For each $n = 1, 11, 21, ..., 141$ we generated each time n random transactions a_i such that $1 \leq s_i \leq 200$ and $1 \leq f_i \leq 100$. $W = 10000$. The dependencies are also generated randomly such that there are no circular dependencies. The data points are calculated as the average of 10 runs. If $|V_a^C| > 15$ it is reduced to 15 using the fee criterion.

The dependency knapsack problem

Experiment II

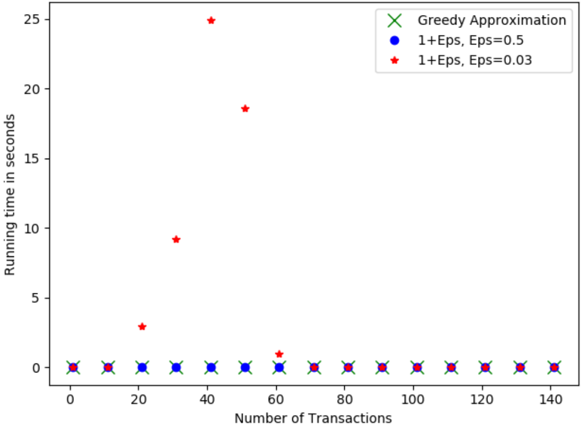


Figure: Runtime in seconds as a function of n

The dependency knapsack problem

Experiment II

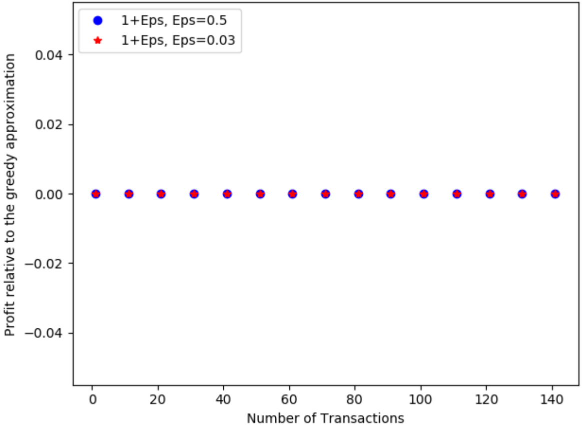


Figure: Profit relative to the greedy approximation as a function of n

Real data

- We checked the dependency knapsack approximations that we implemented on real data.
- The data was sampled using the Bitcoin API.
- Every day, the first sample $t = 1$ returns the whole mempool.
- Other samples $t > 1$ through out the day return two sections:
  - An added section that contains the transactions that are added in the mempool after the last sample at $t - 1$.
  - The added transactions can be dependent upon transactions that were in the mempool at time $t - 1$ but not vice verse.
  - A removed section that contains the transactions that are removed from the mempool after the last sample at $t - 1$.
  - The removed section is usually empty.

Experiment I

First we explore how the approximations behave, especially how the $(1+\varepsilon)$ approximation behaves for different ε. In this experiment we tried

$$\varepsilon \in \{0.01, 0.03, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$$

The data was collected on 28/08/2017 and the time sample is $t = 1$.

Real Data

Experiment I

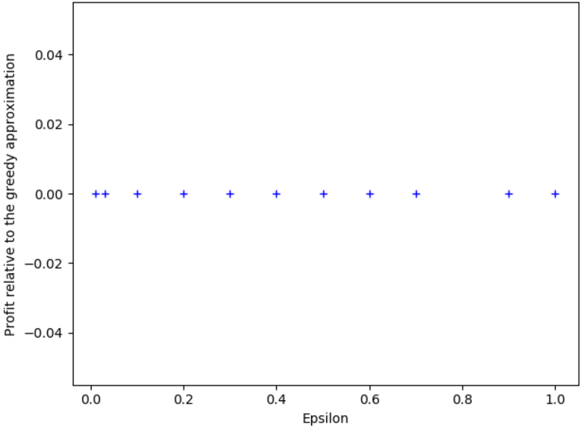


Figure: Profit relative to the greedy approximation as a function of ε

Real Data

Experiment I

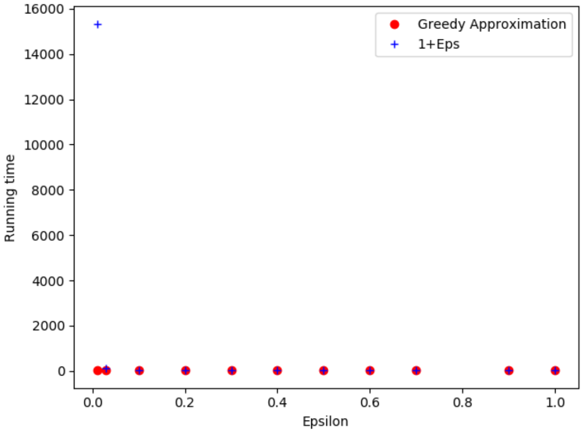


Figure: Runtime in seconds as a function of ε

Real Data

Experiment II

- We saw in the previous experiment that ε had to be really small scuh that $|V_a^C| \neq \emptyset$.
- We explore the greedy approximation and $(1+\varepsilon)$ approximation for $\varepsilon \in \{0.03, 0.1\}$ through out different days.
- The data was sampled at different days at time sample $t = 1$.
- Furthermore, if $|V_a^C| > 10$ it is reduced to 10 using the fee criterion.

Real Data

Experiment II

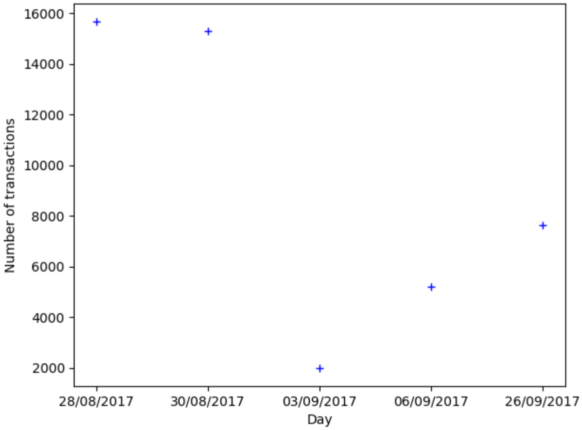


Figure: n at $t = 1$ per day

Real Data

Experiment II

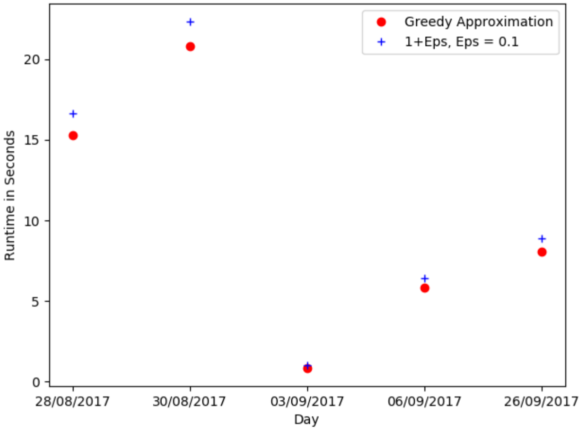


Figure: Runtime in seconds per day

Real Data

Experiment II

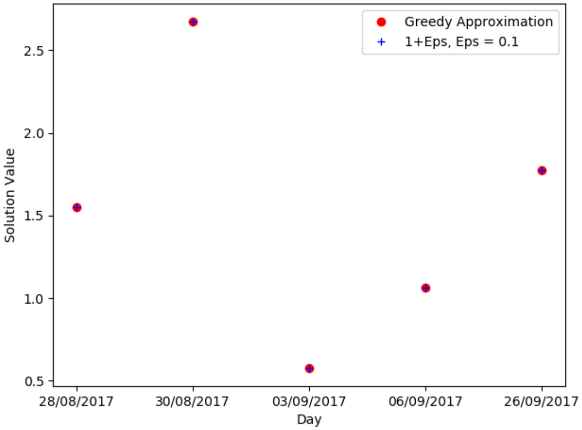


Figure: The solution value per day

Real Data

Experiment II

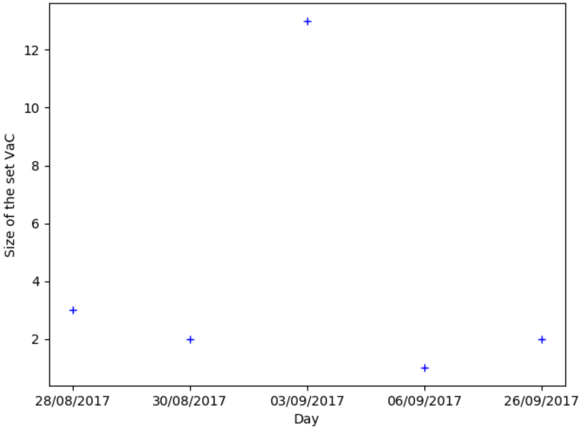


Figure: $|V_a^C|$ per day when $\varepsilon = 0.03$

Real Data

Experiment II

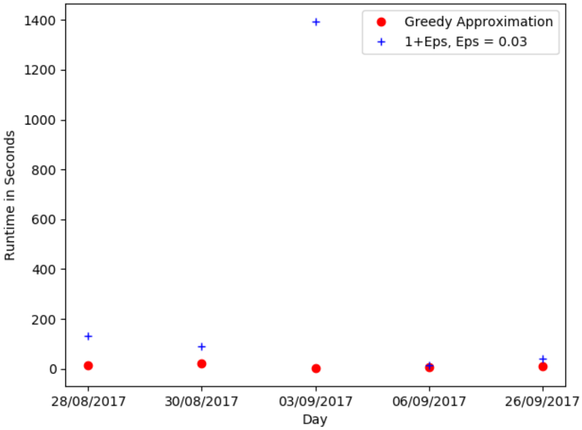


Figure: Runtime in seconds per day

Real Data

Experiment II

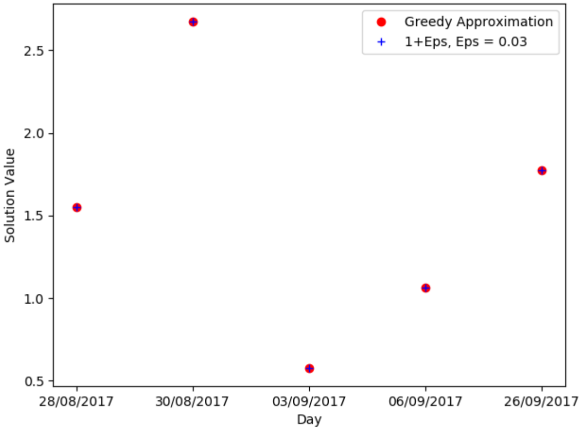


Figure: The solution value per day

Real Data

Experiment III

- We want to explore how the profit changes through out the day.
- We mentioned that when we sample, the removed section is usually empty.
- As such we expect the profit to increase.

Experiment III

We explore the profit as a function of t. The data was sampled on 28/08/2017.

Real Data

Experiment III

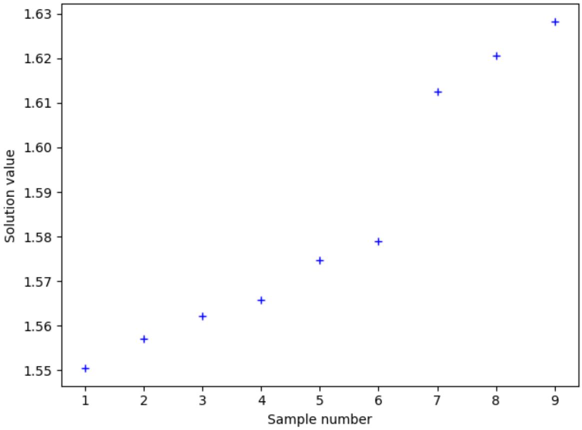


Figure: Solution value as a function of t.

The incremental solution

Mock Data - Experiment I

- We want to compare the two versions of the greedy approximation.
- Incremental solution vs Non-Incremental solution.
- First we analyze them on mock data.

Experiment I

For each $1 \leq n \leq 30$ we create a random instance of a dependency knapsack problem with 100 transactions a_i such that $1 \leq s_i, f_i \leq 100$ and $W = 50000$. Then we add n transactions to this instance such that there are no circular dependencies. The data points are the average of 25 runs.

The incremental solution

Mock Data - Experiment I

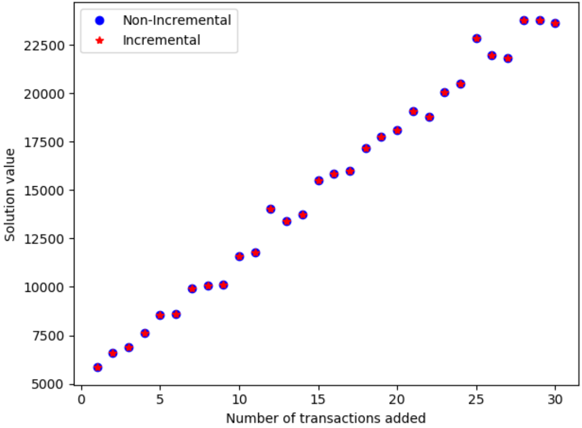


Figure: Solution value as a function of the number of transactions added

The incremental solution

Mock Data - Experiment I

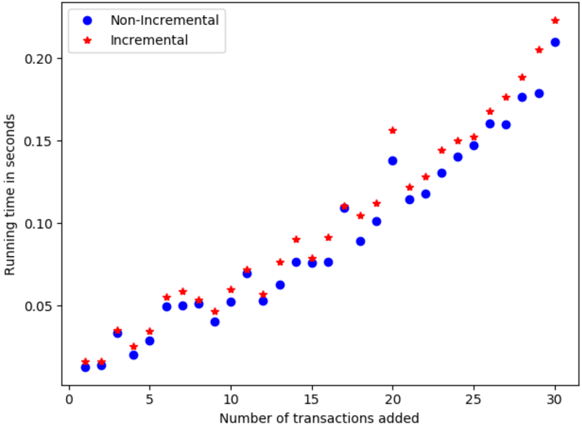


Figure: Running time in seconds as a function of the number of transactions added

The incremental solution

Mock data - Experiment II

Experiment II

We run the same experiment, only that now $n = 1, 11, 21, ..., 291$.

The incremental solution

Mock Data - Experiment II

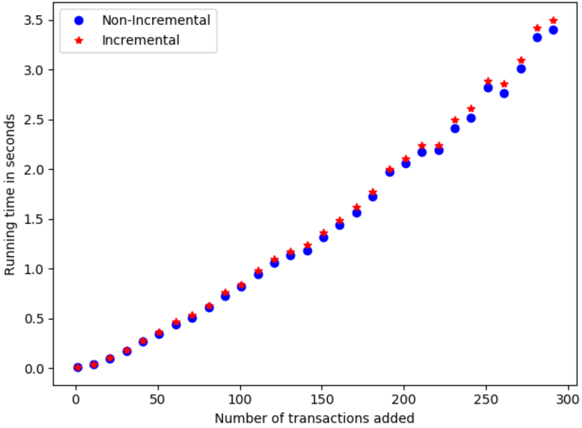


Figure: Running time in seconds as a function of the number of transactions added

The incremental solution

Real Data - Experiment III

- We saw in mock data that the incremental solution doesn't improve the running time of the non-incremental solution.
- This is because in our experiments only a small part of the previous solution became part of the new solution.
- We want to run the incremental solution on real data.
- The real data provides the constraints needed in order to use the incremental solution, since most of the time there are no removed transactions but only added ones.

Experiment III

This data was sampled on 28/08/2017.

The incremental solution

Real Data - Experiment III

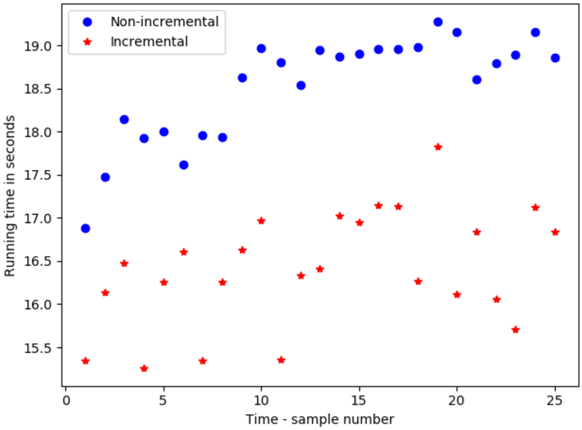


Figure: Runtime in seconds as a function of t

The incremental solution

Real Data - Experiment III

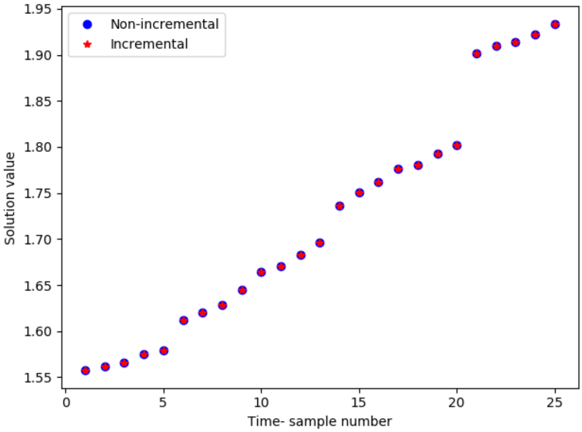


Figure: Solution value as a function of t

The incremental solution

Real Data - Experiment IV

- We saw that on real data the incremental solution makes a difference.
- We get the same solution and save some time.

Experiment IV

Just to make sure we ran the same experiment on the data that was collected on 03/09/2017.

The incremental solution

Real Data - Experiment IV

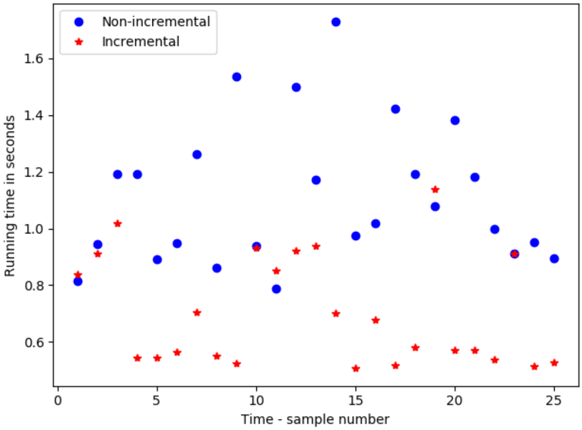


Figure: Runtime in seconds as a function of t

The incremental solution

Real Data - Experiment IV

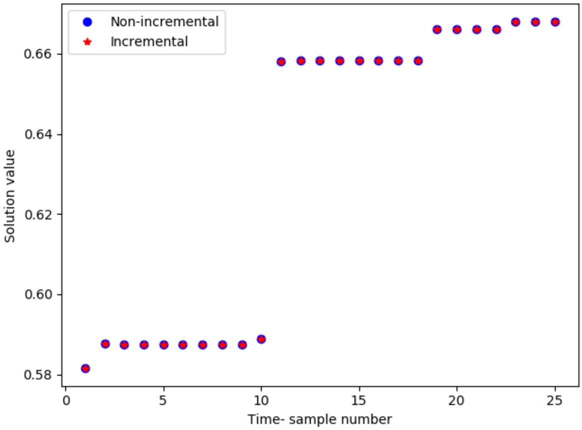


Figure: Solution value as a function of t

Outline

1. Cryptocurrencies
2. The knapsack and dependency knapsack problems
3. Solutions
4. Implementation
5. Results
6. Conclusions

Conclusions

- The $(1+\varepsilon)$ approximation is not feasible because the greedy approximation value $\gg$ fees of transactions. This means that $\varepsilon \ll 1$ which means that $\frac{2}{\varepsilon}$ is big, hence making the algorithm infesiable in terms of time and memory.
- The incremental solution algorithm on real data gives the same results as the greedy approximation algorithm on real data and is faster most of the time. It is important to remember that the incremental solution can be used only if no transactions were removed.