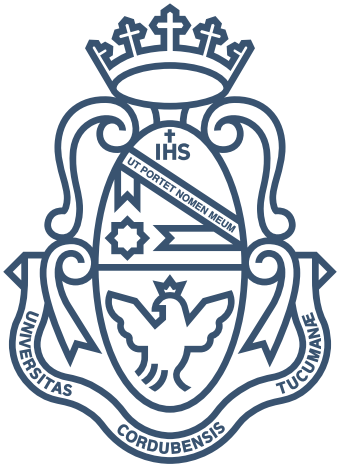
Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía, Física y Computación



Trabajo Especial de la Licenciatura en Ciencias de la Computación

Análisis del desempeño de algoritmos aplicados a la selección de monedas sobre Bitcoin

Directores:
Dr. Luis Ariel Biedma
Dr. Gabriel Eduardo Moyano

Candidato:
César Álvarez Vallero

Año Académico 2023

Agradecimientos

A mi familia por ser mi primer soporte.
A mis tías por el apoyo en todo mi trayecto académico.
A mis amigos y compañeros, por las risas y los ánimos.
A mis directores, por la paciencia a lo largo del desarrollo de este trabajo.

Resumen

Esta tesis propone un análisis comparado de algoritmos de selección de monedas en Bitcoin, nuevos y existentes, a partir de su modelado como problemas de Optimización Lineal Binaria y la simulación de su comportamiento en distintos escenarios buscando encontrar al que mejor optimiza este proceso de acuerdo a los criterios de minimización de costos y de espacio. El desarrollo implica la definición de conceptos relacionados con la implementación del protocolo Bitcoin, la recolección de datos de su red, la formulación de los modelos y la implementación del simulador, que permite la aplicación de los modelos a los escenarios creados a partir de los datos. Los resultados son analizados de forma comparativa teniendo en cuenta los criterios de costo y espacio y cualquier aspecto relacionado a ellos. Finalmente damos nuestra conclusión a partir de lo observado y orientamos a los interesados sobre posibles áreas de trabajo futuro.

Summary

In this thesis, we propose a comparative analysis of new and already-existent coin selection algorithms applied to Bitcoin, modeling them as Binary Integer Programming Problems and simulating their behaviour in different scenarios looking for the one that best optimizes the process regarding minimization cost and space criteria. Its development implies the definition of Bitcoin's protocol-related concepts, the collection of data from its network, the formulation of the models, and the implementation of the simulator, which allows the application of the models in the scenarios derived from the collected data. The results are analyzed comparatively taking into account the cost and space criteria and any other aspect that may influence them. Finally, we present the conclusion reached from the observations and orientate the interested ones about possible areas of future work.

Índice general

1 Introducción

1.1 Introducción

El 31 de octubre de 2008, se hizo público el documento con el título de *Bitcoin: A Peer-to-Peer Electronic Cash System*[1], realizado por un individuo o grupo de ellos bajo el seudónimo de Satoshi Nakamoto. Este documento define un protocolo descentralizado de intercambio de valor, que no requiere de la confianza entre partes o con terceros para su funcionamiento. El 3 de enero de 2009, se puso en funcionamiento la implementación de referencia de este protocolo, *Bitcoin Core*.

Para permitir el intercambio de valor y la contabilización del mismo, el protocolo definió una unidad, a la que se identifica con el mismo nombre del protocolo, bitcoin (BTC). Esta unidad puede subdividirse en cien millones de partes, conocidas habitualmente como satoshis (SATS). Es decir: 1 BTC = $1e8$ SATS. La posesión de un subconjunto de estas unidades esta atada a la resolución de un problema criptográfico. Poseer la solución del problema es equivalente a tener el control de sus unidades asociadas. Tanto el número de unidades como el problema se almacenan de forma conjunta en una estructura de datos llamada *Unspent Transaction Output* o *UTxO*, es decir, las salidas no gastadas de una transacción.

La transferencia de las unidades de un *UTxO* se realiza cuando se suministran las claves que permiten resolver el problema inscripto en este. Llamaremos a este proceso, **desbloqueo del *UTxO***. Estas claves son también utilizadas para generar nuevos y distintos problemas criptográficos, que codificados de acuerdo a formatos predefinidos, permiten el bloqueo de unidades de bitcoin en nuevos *UTxOs*.

Cualquier elemento que permita almacenar estas claves es denominado **Wallet**, o, en español, **Monedero**. Esta puede ser desde una hoja de papel hasta un dispositivo de hardware especializado, cada una con su respectivo balance entre conveniencia y seguridad.

Una transacción se comporta como una función que consume la solución a un problema criptográfico almacenado dentro de un *UTxO*, y crea nuevos *UTxOs* a partir de la resignación de una parte de las unidades desbloqueadas. Tanto la cantidad de entradas como la cantidad de salidas pueden ser mayor o iguales a uno, siempre dentro de los límites consensuados por las reglas del protocolo. La diferencia entre la suma de las unidades consumidas y la suma de las unidades asignadas, conocida como **tarifa de la transacción**, esta destinada a incentivar la incorporación de la nueva transacción en la base de datos del protocolo. En la fig. 1 se puede observar un esquema de este comportamiento, así como la conexión que establecen los *UTxOs* entre las transacciones que los producen y las que los consumen.

La unidad de almacenamiento de esta base de datos son bloques de transacciones. Cada bloque contiene una cantidad finita de ellas, junto con meta datos que permiten asegurar su consistencia y validez en el marco del protocolo y establecen una relación irrepetible entre un bloque y el siguiente, formando una cadena en la cual, la modifi-

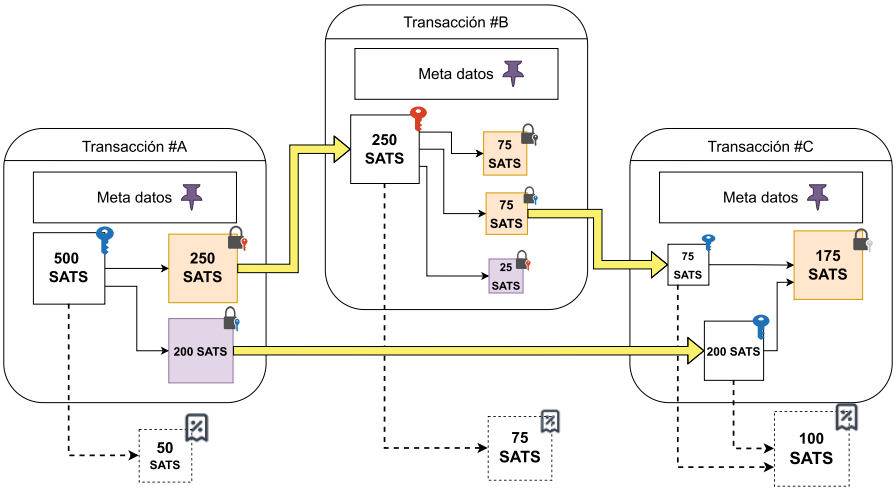


Figura 1: Esquema del intercambio de unidades de bitcoin mediante transacciones.

Los *UTxOs* en violeta son salidas de cambio, los amarillos son salidas de pago y los que no están coloreados y se encuentran en el lado derecho de las transacciones son las entradas de las mismas. El bloque externo con líneas punteadas representa la tarifa que se paga por transacción a los mineros. Las llaves coloreadas y los candados correspondientes representas los programas de bloqueo y las claves de desbloqueo asociadas a los *UTxOs*. Los meta datos de una transacción son elementos necesarios y fijos para asegurar el funcionamiento del protocolo. Las flechas en amarillo fuerte representan las relaciones que se establecen entre transacciones. Notar que estas son relaciones abstractas que no se corresponden con estructuras de datos en el protocolo.

cación de un bloque a una altura arbitraria, implica la modificación de todos los bloques posteriores. Este tipo de base de datos, representada en la fig. 2, se conoce con el nombre de **cadena de bloques** o **blockchain**.

Como la modificación en los datos históricos de la cadena implicaría fraudes, perdidas económicas y la perdida de utilidad del protocolo, este se diseño con un sistema de incentivos que busca alinear los intereses individuales de todos los participantes y desalentar cualquier acción antagónica. Este mecanismo, conocido como **prueba de trabajo**, **Proof of Work** o **PoW**, similar a un sorteo de lotería, requiere la inversión en recursos de computo para participar, pero otorga recompensas comparables en bitcoins. El ganador de este proceso probabilístico obtiene la recompensa y produce el siguiente bloque en la base de datos. A los participantes de esta tarea se los conoce como **mineros**.

Las demoras de red, intentos de ataques al protocolo o la coincidencia hace que en muchas ocasiones coexista más de una cadena de bloques al mismo tiempo. Gracias a que la prueba de trabajo es cuantificable, el protocolo esta diseñado para solo considerar la cadena con mayor prueba de trabajo acumulada. A partir de aquí, cuando nos refiramos a la cadena de bloques, estaremos considerando únicamente aquella cadena donde la prueba de trabajo es mayor, salvo que aclaremos lo contrario.

Cuando una transacción es inscripta dentro del último bloque de la cadena, se dice que **la transacción ha sido confirmada**. A medida que se escriben más bloques en la cadena, la transacción acumula más confirmaciones.

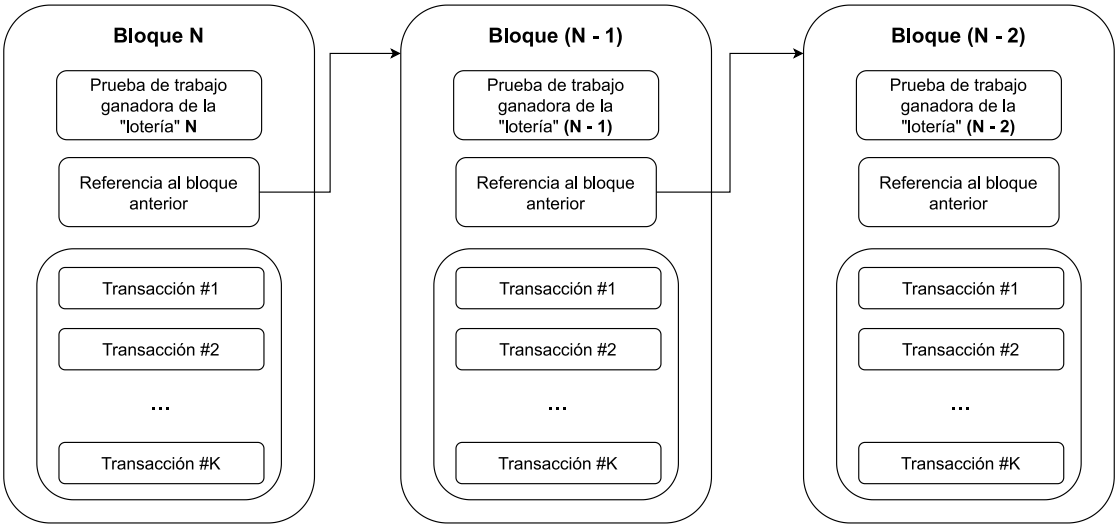


Figura 2: Esquema de una cadena de bloques

Dada la naturaleza estocástica del mecanismo, siempre existe la posibilidad de modificación de la cadena a partir de una altura determinada, con cierta probabilidad de éxito. Es por eso que en general no se considera un *UTxO* como final hasta que la transacción que lo ha generado recibe 6 confirmaciones. A partir de esta cantidad de bloques la modificación se vuelve improbable, siempre y cuando los incentivos se mantengan equilibrados.

Cada bloque puede almacenar una cantidad finita de transacciones y los mineros son libres de seleccionar el conjunto de estas más conveniente a su tarea, es por ello que existe un mercado por el espacio disponible en cada bloque. Este mercado utiliza el excedente o tarifa de la transacción, para pujar y ganar un espacio en la cadena dentro de un determinado número de bloques. De acuerdo a la urgencia que tenga el emisor de la transacción por confirmarla en la cadena, será el excedente que pagara a los mineros. Todo participante racional de este mercado buscara pagar la cantidad justa para cumplir su objetivo, minimizando en lo posible el costo de alcanzarlo.

La realización de una transacción involucra entonces la selección de un número de *UTxOs* que acumulen suficiente cantidad de bitcoin como para satisfacer el pago y sus costos, minimizándolos en lo posible. Dichos costos son expresados de forma directa a través de las tarifas pagadas a los mineros, pero pueden tomar otras formas que veremos más adelante. Se conoce a este proceso como **selección de monedas** o **coin selection**.

En el presente trabajo buscamos aislar el proceso de selección de monedas de los demás componentes del protocolo y mediante simulaciones hacer un análisis comparado del comportamiento de distintos algoritmos aplicados al problema.

1.2 Motivación

De acuerdo a lo desarrollado las transacciones en Bitcoin consumen y producen nuevos *UTxOs*. La existencia de tarifas, incentivan a los usuarios que realizan transacciones a minimizarlas. Los usuario disponen de un número finito de *UTxOs*, con una cantidad de bitcoin bloqueada en ellos y como toda estructura de datos ocupa un espacio en memoria, a partir del cuál deriva el valor de las tarifas.

Aunque la influencia de las tarifas es de primer orden existen otros factores que influyen de forma más sutil, por lo que no se ha analizado su impacto en profundidad.

Un factor a considerar al momento de realizar una transacción, es la privacidad. Uno de los objetivos del protocolo es que el intercambio de valor no sea censurable. Cualquier vulneración en la privacidad de los usuarios podría implicar una limitación al momento de transaccionar. En *Bitcoin Core* se han implementado heurísticas orientadas ha minimizar este problema[1], pero no existe aún un algoritmo que mejore la selección de acuerdo a criterios de privacidad.

Existen mecanismos alternativos que permiten aumentar la privacidad del conjunto de *UTxOs*, como CoinJoin[2] o CoinSwap[3], aunque la eficacia de estos depende de la disciplina del usuario y en algunos casos se ha visto vulnerada por malas prácticas de los mismos o fallos en la implementación[4].

El análisis de la efectividad de estas tecnologías a llevado a la elaboración de métricas, como la **entropía de una transacción**[2], que permiten ponderar la privacidad de una transacción a través de un número. Su cómputo requiere el cálculo de probabilidades de acuerdo a un conjunto de combinaciones entre las entradas y salidas de la transacción, un problema adicional, perteneciente al conjunto de los NP-Hard, por lo que para el presente trabajo hemos descartado su incorporación.

Otro aspecto influyente en el costo es la huella en memoria a nivel de la red de nodos. Para aumentar la velocidad de procesamiento de las transacciones, cada nodo de la red de Bitcoin mantiene en memoria un conjunto de los *UTxOs*, es decir, las salidas de transacciones que aún no han sido consumidas por ninguna otra transacción. A medida que se procesan transacciones, los nodos actualizan este conjunto. El balance entre el número de *UTxOs* consumidos y generados por una transacción determina que este conjunto crezca o decrezca. Si no se lo tiene en cuenta, su aumento no controlado produciría una presión contra los nodos de la red más restringidos a nivel de hardware, atacando la descentralización de la misma. Desde los inicios del protocolo, el crecimiento del conjunto global de *UTxOs* no ha desacelerado su crecimiento (fig. 3). Este problema ha sido señalado previamente[3] y en algunos caso se han propuesto soluciones, relacionada

[1] josibake. *wallet: avoid mixing different OutputTypes during coin selection.* Github Pull Request. May 2022. URL: `https://github.com/bitcoin/bitcoin/pull/24584`.

[2] *CoinJoin.* Bitcoin Optech. URL: `https://bitcoinops.org/en/topics/coinjoin/`.

[3] *CoinSwap.* Bitcoin Optech. URL: `https://bitcoinops.org/en/topics/coinswap/`.

[4] Jamie Redman. "Journalist Claims She Identified the 2016 DAO Hacker, Evidence Shows Investigators 'De-Mixed' Wasabi Transactions". In: *Bitcoin.com, Saint Bitts LLC* (Feb. 2022).

por ejemplo, con propiedades de compresibilidad de las transacciones[4].

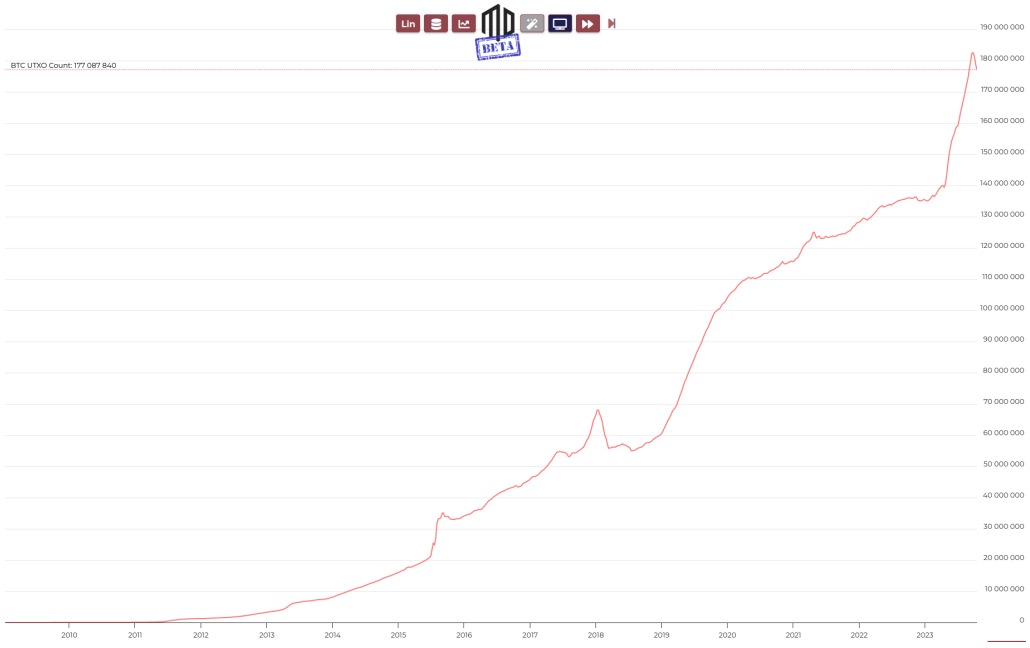


Figura 3: Crecimiento histórico del conjunto de *UTxOs*
Fuente: `https://www.mesmerdata.com/on-chain-charts/btc-utxo-count/`

El crecimiento irrestricto de un conjunto particular de *UTxOs* incrementa el número de variables a considerar para ciertos algoritmos, afectando también en la velocidad de procesamiento. A si mismo, una estrategia demasiado conservadora, podría reducir el número de *UTxOs* a niveles en los que la realización de cualquier transacción revela mas información de la deseable acerca de los balance disponibles o el conjunto carece de la flexibilidad para adaptarse a distintos escenarios, haciendo la realización de transacciones imposible bajo ciertas tarifas.

Este trabajo propone el modelado, análisis y simulación de soluciones al problema de la selección de monedas, enfocándonos en un subconjunto de los criterios de optimización enumerados arriba.

1.3 Esquema de la tesis

Este documento esta estructurado de la siguiente manera: En el Cap. 2 establecemos la teoría que da soporte a nuestro enfoque e introducimos bibliografía relacionados a la selección de monedas. En el Cap. 3, añadimos notación de soporte que nos permitirá definir formalmente el problema y dar los modelos sobre los que fundamentamos el desarrollo práctico de nuestra solución. En el Cap. 4 explicamos algunos aspectos del simulador que acompaña esta tesis, detallando la información extraída por el mismo y los modelos finalmente implementados. Luego desarrollamos los escenarios simulados y

expone los resultados de la simulación, junto con nuestro análisis. En el Cap. 5 cerramos el trabajo dando nuestras conclusiones y posibles líneas de investigación a futuro.

2 Fundamentos y trabajos relacionados

En este capítulo, primero hablaremos del campo teórico y la perspectiva de análisis a partir de la cual modelaremos el problema de la selección de monedas. Luego introduciremos trabajos relacionados en el área y por ultimo destacaremos los componentes más influyentes en nuestro enfoque.

2.1 Fundamentos

La selección de monedas en bitcoin es el problema de optimización en el cual se escogen los *UTxOs* que formaran parte del conjunto de entradas de una transacción. Para ello se calcula una tarifa por unidad de peso de acuerdo a las condiciones del mercado del espacio de bloques y la urgencia por colocar dicha transacción dentro de los siguientes n bloques. El objetivo es lograr su confirmación en el número deseado de bloques minimizando los costos de esta acción. Dado que no es posible gastar parcialmente un *UTxO*, la tarea es propensa a generar excedentes, los cuales pueden retornarse al usuario creando un nuevo *UTxO* como salida de cambio de la transacción. Este nuevo *UTxO* también debe pagar tarifas, por lo que se debe considerar si el excedente producido amerita el nuevo gasto, o es más beneficiosa su agregación a la tarifa.

Existen otras consideraciones respecto de la creación de una transacción, como por ejemplo la eliminación de vínculos entre transacciones[5] o la reducción del conjunto de *UTxOs*[3]. En este trabajo nos centraremos en dos, la minimización de las tarifas y el control del conjunto de *UTxOs*.

Este problema esta relacionado con el *subset sum problem*. El mismo es un problema de decisión definido a partir de un conjunto numérico $W = w_1, ..., w_n \subset \mathbb{Z}$ y una suma $T \in \mathbb{Z}$, donde el objetivo es encontrar el subconjunto $Y \subseteq W$ tal que $\sum_{i=1}^{|Y|} y_i = T$. La pregunta a responder, en este caso, es si existe tal subconjunto.

Debido a que existen 2^n posibles subconjuntos de W, el problema clasifica como NP-Hard. La aplicación de restricciones, heurísticas o variaciones en el tamaño del conjunto o la precisión (numero de bits para representar T) deseada, permiten realizar optimizaciones, pero el carácter exponencial del problema permanece.

Si en lugar de considerar el subconjunto que exactamente suma T, buscamos aquellos que suman al menos T, y sujeto a esto, queremos obtener el subconjunto con la suma lo más cercana a T posible, obtenemos un problema de optimización, también NP-Hard

La selección de monedas se puede definir manera análoga, donde tenemos un conjunto $U = \{u_1, u_2, u_3, ..., u_n\}$ de *UTxOs*, con sus respectivos valores $V = \{v_1, v_2, v_3, ..., v_4\}$ y pesos denominados en la unidad utilizada por el protocolo $P = \{p_1, p_2, p_3, ..., p_n\}$. Una tarifa por unidad de peso objetivo para introducir la transacción en B bloques igual a γ_B, y un conjunto de obligaciones de pago con valores $R = \{r_1, r_2, r_3, ..., r_m\}$ y respectivos pesos $C = \{c_1, c_2, c_3, ..., c_m\}$ asociados a los nuevos *UTxOs* a crear, más un peso constante de transacción T. Si la inclusión de $u_1, u_2, u_3, ..., u_n$ esta determinada por las variables binarias $x_1, x_2, x_3, ..., x_n$, entonces la distancia al costo de satisfacer una

transacción bajo la tarifa γ_B esta definida por:

$$\begin{aligned} \arg\min_i \quad & \sum_{i=1}^{n} x_i \cdot (v_i - p_i \cdot \gamma_B) \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \cdot (v_i - p_i \cdot \gamma_B) \geq \sum_{j=1}^{m} (r_j + c_j \cdot \gamma_B) + T \cdot \gamma_B, \\ & x_i \in \{0,1\} \quad \forall i \in \mathbb{Z}. \end{aligned} \tag{1}$$

La ecuación (1) nos permite formular el problema dentro de los criterios de la *Optimización Lineal*, una rama de la *Optimización Matemática*, donde los problemas se modelan a partir de relaciones lineales, funciones objetivo y restricciones, en la búsqueda del mejor resultado posible. Cuando algunas variables del modelo están restringidas al conjunto de los números enteros, se trata de *Optimización Lineal Entera* o *ILP* por sus siglas en inglés. En este caso se trata de *Optimización Lineal Binaria* debido a que la variable esta restringida al conjunto $\{0, 1\}$. Usualmente se modelan con la siguiente estructura canónica:

$$\begin{aligned} \max \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \\ \text{y} \quad & \mathbf{x} \in \mathbb{Z}^n. \end{aligned}$$

En el caso de los problemas de minimización, se puede modelar y resolverlos transformándolos a un problema de maximización.

Existen múltiples algoritmos y heurísticas orientadas a producir soluciones a partir de modelos de *Optimización Lineal*. El desempeño de las mismas depende fuertemente del problema. En general no existe una sola técnica de resolución y se termina aplicando una combinación de ellas. En el campo de la *Optimización Lineal Entera*, *Applied Mathematical Programming*[6] reúne estas técnicas bajo tres paradigmas:

- técnicas de enumeración
- técnicas de seccionamiento de planos
- técnicas basadas en la teoría de grupos

Nos enfocaremos en la técnica *Branch and Bound* o *Branch and Cut*, debido a que la actual implementación del algoritmo de selección de monedas en *Bitcoin Core* se basa en este paradigma. De tipo enumerativo, aplica una estrategia de *"divide y vencerás"* sobre la región factible, subdividiéndola para encontrar límites superiores e inferiores que permitan acotar el espacio de soluciones, hasta encontrar la óptima o una aproximación. En cada subdivisión puede ocurrir una de las siguientes:

1. La región determinada por la subdivisión no es factible

2. La solución óptima dentro de la región subdividida es entera
3. La solución es la mejor obtenida hasta el momento

En el caso número 1, se descarta la subdivisión. En el caso número 2, se compara con el límite opuesto al sentido de la optimización (en el caso de maximización, el límite inferior) y se lo actualiza. En el caso número 3, se repite el proceso anterior con respecto al límite opuesto (en el caso de maximización, el límite superior). El proceso finaliza cuando los límites coinciden o, en el peor de los casos, se ha realizado una búsqueda exhaustiva en todo el espacio de soluciones. Los criterios de finalización pueden ser alterados, por ejemplo, considerando un tiempo máximo de ejecución o una distancia entre los límites.

Existen múltiples herramientas orientadas a la *Optimización Matemática*, algunas enfocadas en el modelado, como OR-Tools (Google)[5], AMLP[6] o Pulp[7] y otras que implementan distintas variedades de solvers, como Gurobi[8], GLPK (GNU)[9] o Coin-OR[10], por nombrar algunas.

De los mencionados, elegimos Pulp[7] para realizar los modelos y Coin-OR[8] para resolverlos, basando nuestra decisión en la integración preexistente de ambas herramientas, la interfaz en Python, el código abierto y la implementación de *Branch and Bound* dentro de Coin-OR.

2.2 Trabajos relacionados

Una de las primeras referencia a la selección de monedas en el marco de la *Optimización Matemática*, fue realizada en un artículo de la wiki de Gregory Maxwell[9], un contribuyente de *Bitcoin Core*, muy activo en los inicios del protocolo.

En el plantea que la selección de monedas es un problema de *Optimización Matemática No-Lineal Mixto* que se puede convertir a uno *Lineal* si no tenemos en cuenta la estimación de la tarifa como parte del modelo, además sugiere la utilización de técnicas del estilo de *Branch and Bound* para la resolución de los mismos.

Posteriormente, Mark Erhardt en su tesis de maestría[10], pone en duda la actualidad de la aserción de Maxwell sobre la no linealidad de los modelos de selección de monedas debido a cambios posteriores a la misma, que modificaron la forma de seleccionar transacciones por parte de los mineros, afectando al mercado de tarifas subyacente.

En ese mismo trabajo, Erhardt, explica los desafíos que involucran el problema de selección de monedas, modela las restricciones del mismo y lista posibles estrategias para su solución. Siguiendo la recomendación de Maxwell implementa un algoritmo inspirado en la técnica de *Branch and Bound* y mediante simulaciones realiza una evalu-

[5] `https://developers.google.com/optimization`
[6] `https://ampl.com/`
[7] `https://pypi.org/project/PuLP/`
[8] `https://www.gurobi.com/`
[9] `https://www.gnu.org/software/glpk/`
[10] `https://coin-or.github.io/`

ación comparada del desempeño del algoritmo de Bitcoin utilizado a la fecha, su propia implementación y las correspondientes a las otras estrategias analizadas. En este caso no se considero la generación de múltiples salidas de pago en una misma transacción, pero es un escenario posible dentro del protocolo.

Fruto de este trabajo se comenzó a utilizar el algoritmo basado en *Branch and Bound* como estrategia de selección de monedas principal de la implementación matriz del protocolo, *Bitcoin Core*[11].

Posteriormente, Erhardt ideo e introdujo en *Bitcoin Core* una métrica[12][11] que permite calificar una selección de monedas de acuerdo a la diferencia entre la tarifa por unidad de peso al momento de producirse la selección y una tarifa de referencia. Aunque esta métrica ha demostrado ser útil y adaptarse a escenarios variados, bajo condiciones extremas del mercado de tarifas puede producir el agotamiento del conjunto de *UTxOs* del usuario. Sin embargo, por su utilidad para calificar selecciones, la tendremos en cuenta en la elaboración de las funciones objetivo de nuestros modelos.

Dentro de las otras estrategias evaluadas, considera lo que él denomina *"Double Target"*, que consiste en buscar generar siempre una salida de cambio con un valor igual al del pago realizado. Esta estrategia ya había sido señalada por Lopp en un artículo de su blog personal[12].

Los resultados de las simulaciones no validaron la hipótesis de que este tipo de estrategia permita disponer de mayor número de *UTxOs* con un valor cercano a la denominación de las solicitudes de pago más habitual, ni que la estrategia genere mayor número de soluciones exactas. Igualmente, utilizaremos una variante de la misma durante la experimentación, bajo las mismas hipótesis, para atender los casos donde la generación de salidas de cambio es inevitable.

Edsko de Vries, en su artículo, *Self Organization in Coin Selection*[13] enfocado en la criptomoneda Cardano, posterior a Bitcoin, recoje la estrategia de *"Double Target"* de Erhardt y la re introduce en lo que el llama principios de auto organización. Precisamente, en el principio número dos desarrolla la misma hipótesis: Si por cada solicitud de pago por un valor x creamos una salida de cambio cercana al mismo valor x, entonces terminaremos con una abundancia de salidas de cambio en nuestro conjunto de *UTxOs* por un valor x precisamente cuando las solicitudes de pago por dicho valor también abundan.

A partir de este principio, elaboro el algoritmo *Random Improve*, sobre la red de Cardano. La estrategia consiste en incluir una salida de cambio por cada solicitud de pago de aproximadamente el mismo valor. El análisis del desempeño tuvo buenos resultados para Cardano, pero diferencias en el cálculo de tarifas, capitalización de mercado y descentralización entre ambas criptomonedas, hacen que esta estrategia sea demasiado

[11] achow101. *Coin Selection with Murch's algorithm.* Github Pull Request. June 2017. URL: `https://github.com/bitcoin/bitcoin/pull/10637`.

[12] achow101. *wallet: Decide which coin selection solution to use based on waste metric.* Github Pull Request. May 2021. URL: `https://github.com/bitcoin/bitcoin/pull/22009`.

costosa como para aplicarla de forma competitiva en Bitcoin. De agregar una salida de cambio por cada solicitud de pago la tarifa total de las salidas de cada transacción al menos se duplicaría. Una alternativa es considerar una única salida de cambio con un valor cercano al acumulado de todas las solicitudes de pago.

La implementación del algoritmo de selección de monedas previa al trabajo de Erhardt[13], estaba orientada a encontrar un subconjunto de *UTxOs* que permitiera cubrir de forma exacta el monto acumulado de las solicitudes de pago junto con las tarifas de la transacción. En caso de no ser posible, variando el conjunto de selección, se ejecutaban múltiples rondas de un *Knapsack Solver*, en busca de minimizar la creación de salidas de cambio con un valor de base de $1e6$ satoshis.

Esta implementación permitía mantener el conjunto de *UTxOs* en un tamaño razonable y generaba soluciones de forma pseudo aleatoria, minimizando la trazabilidad de la transacción basándose en criterios propios de los *UTxOs*. Sin embargo, era propensa a elevar las tarifas de la transacción, computacionalmente intensiva (1000 iteraciones por cada ejecución del *Knapsack Solver*), la generación de cambio era fácilmente asociable a este algoritmo debido a su denominación de base ($1e6$ satoshis) y la consolidación agresiva de los *UTxOs* tendía a incluir aquellos cuya tarifa superaba el valor del mismo a la tarifa por unidad de peso del momento.

Buscando mejorar *Knapsack Solver*, el trabajo independiente de Daniel Diroff[14] introdujo el algoritmo de selección de monedas *Coin Selection with Leverage*. El mismo modela el problema nuevamente desde el enfoque de la *Optimización Matemática*, implementando el algoritmo a través de la combinación de tres modelos distintos. El primero busca obtener una solución libre de cambio, el segundo, encadenar dos transacciones de forma tal que el cambio de una sea entrada de la siguiente (notar que ambas opciones buscan evitar la generación de salidas de cambio). Finalmente, el tercero busca una selección considerando la creación de salidas de cambio.

El trabajo de Diroff estaba orientado a escenarios donde la recepción de solicitudes de pago permitía este encadenamiento, como casas de cambio o procesadores de pago. Los resultados obtenidos a partir de las simulaciones arrojaban buenos márgenes de ahorro para transacciones con números de solicitudes de pago mayores o iguales a 2, respecto del *Knapsack Solver* utilizado hasta ese momento en *Bitcoin Core*. No nos consta que se este utilizando actualmente en ningún prestador de los servicios mencionados.

Tanto el trabajo de Erhardt como el de Diroff incluían sendos programas para simular los algoritmos implementados. El de Erhardt, programado en Scala[15], realizaba la implementación de los algoritmos de forma manual. El trabajo de Diroff en cambio, fue programado en Python[16], utilizando la interfaz Pulp sobre la implementación *Branch and Bound* de *Coin-OR*, *Coin-OR/CBC*.

En ambos trabajos se realizo una representación menos compleja de la implementación original del protocolo. Esto permite aislar cualquier demora asociada a procesos que for-

[13] *Bitcoin Coin Selection Algorithm*. Bitcoin Core v0.16.3. Sept. 2018. URL: `https://github.com/bitcoin/bitcoin/blob/v0.16.3/src/wallet/wallet.cpp#L2384`.

man parte del mismo pero no relacionadas al problema de la selección de monedas, como la generación de bloques, o la transmisión de la transacción a través de la red.

En un trabajo posterior, Erhardt junto con Andrew Chow, el desarrollador a cargo del mantenimiento de la **Wallet** de *Bitcoin Core* implementaron un simulador programado en Python pero utilizando una red de pruebas provista por Bitcoin[17]. En ese simulador se encuentran cargados distintos escenarios adaptados a partir del trabajo original de Erhardt junto con variaciones posteriores o escenarios propios generados a partir del comportamiento de selecciones reales.

Este trabajo se inspira en el desarrollo de Diroff para implementar un simulador de algoritmos de selección de monedas en Python utilizando el enfoque de la *Optimización Lineal Binaria* para el modelado de los objetivos y restricciones del problema mediante la librería *Pulp* y su interfaz a la implementación del algoritmo *Branch and Bound* provista por *Coin-OR*, para la resolución de estos modelos. Para la elaboración de los algoritmos se consideraran tanto estrategias básicas (aleatorias o *greedy*) así como modelos más complejos elaborados a partir del análisis algorítmico y las métricas surgidas de los trabajos de Erhardt y De Vries. Como escenarios de pruebas utilizaremos los suministrados por Erhardt y Chow en la implementación de su simulador en Python.

3 Metodología

Para dar soluciones al problema de la selección de monedas, desde la óptica de la *Optimización Lineal Binaria*, elaboraremos una serie de modelos formados por funciones objetivo y restricciones a dicha función. Luego programaremos estos modelos mediante la herramienta *Pulp* en Python y finalmente realizaremos un análisis comparado de los resultados.

Comenzaremos estableciendo la notación, continuaremos con constantes y definiciones de soporte y finalmente daremos los modelos finales a simular.

3.1 Notación complementaria

Las siguientes definiciones son copias textuales del apunte de Lógica y Computabilidad de la carrera de Licenciatura en Ciencias de la Computación de la Facultad de Matemática, Astronomía, Física y Computación de la Universidad Nacional de Córdoba[18], y nos permitirán definir de forma exacta algunos componentes de nuestro marco teórico.

Definición 1 (Restricción al dominio de una función)**.** *Dada una función f y un conjunto $S \subseteq D_f$, usaremos $f|_S$ para denotar la restricción de f al conjunto S, i.e. $f|_S = f \cap (S \times I_f)$. Nótese que $f|_S$ es la función dada por*

$$\begin{aligned} D_{f|_S} &= S \\ f|_S(e) &= f(e), \text{ para cada } e \in S \end{aligned}$$

Cualesquiera sea la función f tenemos que $f|_\emptyset = \emptyset$ y $f|_{D_f} = f$.

Definición 2 (Proyección de una tupla)**.** *Dados conjuntos cualesquiera $X_1, X_2, ..., X_n$ con $n \in \mathbb{N}_0$, definimos la siguiente función:*

$$\begin{aligned} \pi_k : X_1 \times ... \times X_k \times ... \times X_n &\to X_k \\ \pi_k(x_1, ..., x_k, ..., x_n) &\to x_k \end{aligned}$$

Llamaremos a $\pi_k(t)$ la proyección del k-ésimo elemento de t.

Definición 3 (Partición)**.** *Dado un conjunto A por una partición de A entenderemos un conjunto $\mathcal{P}$ tal que:*

- *Cada elemento de $\mathcal{P}$ es un subconjunto no vacío de A*
- *Si $S_1, S_2 \in \mathcal{P}$ y $S_1 \neq S_2$, entonces $S_1 \cap S_2 = \emptyset$*
- *$A = \{a : a \in S$, para algún $S \in \mathcal{P}\}$*

Notación 1. *Dado un conjunto A utilizaremos la notación $\{A\}$ en el contexto del tipo de una función, para indicar que la función recibe un conjunto de elementos de A como argumento.*

3.2 Definiciones

Con el objetivo de delimitar nuestro problema de forma precisa, sin sobre simplificar, pero con la suficiente generalidad para que nuestra solución sea flexible, definiremos los términos y conceptos sobre los que basaremos la definición de nuestro problema y su solución, inspirándonos en el trabajo de Atzei, Bartoletti, Lande y Zunino, *A Formal Model of Bitcoin Transactions*[19].

Los elementos a continuación explicados se desprenden directamente del diseño y las estructuras definidas en el protocolo. La decisión de no hacer uso directo de las mismas se basa en la complejidad de las originales, y a que algunos de sus componentes pueden ser ignorados para explicar la selección de monedas.

Definición 4 (Tipo de salida)**.** *Identificador del tipo de programa que permite establecer un mapeo entre una una clave criptográfica y una cantidad finita de bitcoin. Denotaremos al conjunto de estos con Type.*

En Bitcoin se utilizan distintos formatos para representar direcciones a las que se pueden enviar bitcoins. Dichos formatos de dirección son codificaciones derivadas de la clave pública de un esquema criptográfico asimétrico y son predefinidas y consensuados por los participantes del protocolo. Es su contra parte privada la que permite hacer uso de los bitcoins bloqueados en estas direcciones. Algunas de estos formatos son: *Pay to Public Key* (P2PK), *Pay to Public Key Hash* (P2PKH), *Pay to Witness Public Key Hash* (P2WPKH), *Pay to Taproot* (P2TR) o *Pay to Witness Script Hash* (P2WSH). Con *Type* hacemos referencia al conjunto de todas ellas, aunque a nivel de simulación solo implementaremos P2WPKH, por ser la de uso mas extendido y para simplificar el código.

Definición 5 (Prueba de verificación)**.** *Cadena de caracteres derivada de la clave privada de un esquema criptográfico (proof) correspondiente con cierto tipo de salida (Type) que certifica la propiedad de una cantidad de bitcoin. Identificaremos con Proof al conjunto de ellas, definido de la siguiente manera:*

$$Proof = \bigcup_{k \in Type} Proof_k$$
$$s.t. \quad Proof_k = \{proof_i : i \in \mathbb{N}_0\}$$

Como mencionábamos en el párrafo anterior, los esquemas criptográficos asimétricos tienen una componente pública y una contra parte privada. En Bitcoin, esta componente privada se utiliza para derivar una cadena de caracteres llamada **signature** o **firma**, que demuestra la correspondencia con la clave pública a la que están vinculados los bitcoins. En nuestro trabajo, representaremos dicha **firma** con $Proof$.

Definición 6 (Script de bloqueo)**.** *Denotaremos con Script a toda función del siguiente tipo:*

$$Script : Proof \quad \rightarrow \quad \{true, false\}$$

Las transacciones en Bitcoin no requieren interactividad entre los interesados, por lo tanto, la forma de ejecutar la transferencia de valor entre ambas partes es mediante un programa[14] que verifique la correspondencia entre la **firma** presentada con la dirección a la que se mueven los fondos. Para ello, Bitcoin fue diseñado con un lenguaje de programación, no Turing completo, para poder programar distintas formas de bloquear cantidades de Bitcoin. Cuando nos referimos a un elemento del conjunto $Type$, en la implementación real del protocolo, estos identifican programas predefinidos de diversa complejidad, que contienen la lógica para verificar esta correspondencia, arrojando una respuesta de verdadero o falso de acuerdo a si la **firma** suministrada fue generada por la clave privada correcta o no. Luego son codificados en formatos serializables para convertirse en las direcciones[15] de Bitcoin.

El conjunto *Script* refiere a todos estos programas.

Definición 7 (Salidas de una transacción)**.** *Denotaremos por TxO al siguiente conjunto de 2-uplas:*

$$TxO = \{(Script|_k, v) : k \in Type, 0 \leq v \leq 21 \times 10^{14}\}$$

Llamaremos a TxO el conjunto de salidas de una transacción.

Sin entrar en los detalles de como se generan los elementos de TxO, es importante notar aquí que la forma de circular de los bitcoins es en estructuras que asocian un programa con una cantidad de bitcoin, y uno posee esta cantidad de bitcoin cuando **posee la solución del programa**. Aquí no existe el modelo habitual de cuenta, al que estamos acostumbrados en los bancos o sistemas financieros tradicionales. Todos los bitcoins en circulación están repartidos en estructuras del estilo de las expresadas arriba.

Dado el conjunto TxO definimos la siguiente partición: $\{UTxO, STxO, NoTxO\}$.

Donde $UTxO$ es el conjunto de *Unspent Transaction Outputs*, es decir, aquellos TxO

[14] `https://en.bitcoin.it/wiki/Script`
[15] `https://en.bitcoin.it/wiki/Invoice_address`

que pertenecen al conjunto de salidas de alguna transacción almacenada en la cadena de bloques, pero no pertenecen al conjunto de entrada de ninguna de ellas. Aquellas que si lo hacen forman parte de $STxO$. Luego existe el complemento de estos dos conjuntos, $NoTxO$, donde conviven todas las posibles combinaciones de *Scripts* y valores.

Dentro de $NoTxO$ las reglas del protocolo dejan fuera muchas de las posibles combinaciones. Para simplificar, cuando nos refiramos a elementos de este conjunto, siempre hablaremos de aquellas combinaciones que no rompen ninguna regla consensuada en el protocolo.

La transición de uno a otro conjunto es ordenada y se da siempre de la misma manera. Un elemento de $NoTxO$ solo puede pasar a pertenecer a $UTxO$ si cumple las reglas de consenso. Una ves dentro del conjunto de salidas de una transacción, no puede volver a $NoTxO$. Respectivamente, cuando se gasta dicho $UTxO$, pasa irreversiblemente al conjunto de $STxO$.

Definición 8 (Monedero)**.** *Llamaremos monedero a cualquier conjunto* $W \subseteq Proof$.

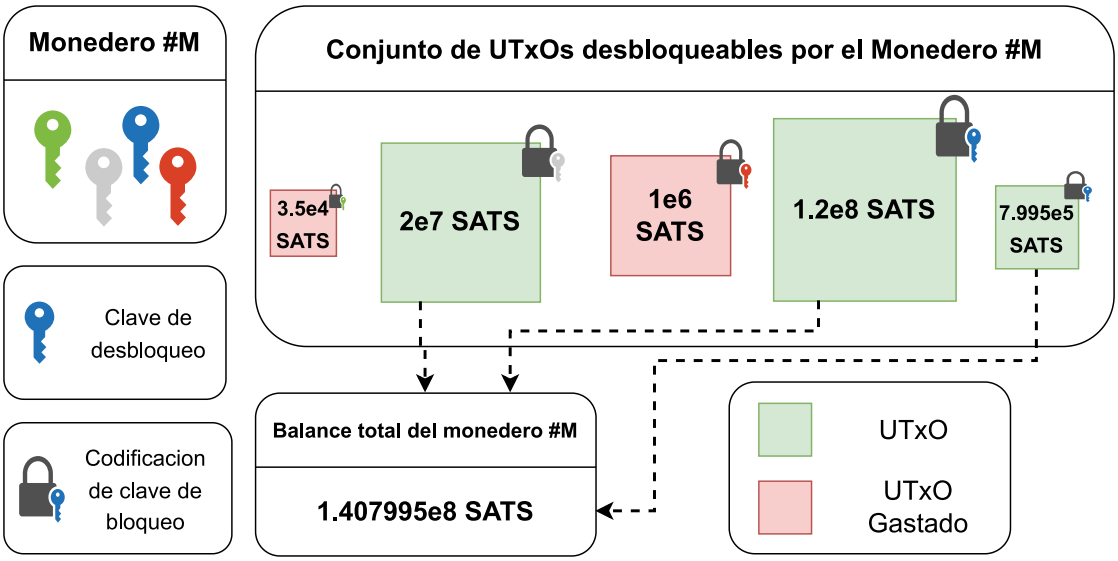


Figura 4: Esquema de un monedero

De esta manera nos reducimos a la mínima expresión de un *monedero* en Bitcoin, esto es, el conjunto de claves asociadas a $TxOs$ mediante un *Script* (fig. 4).

Dado un *monedero* W, podemos identificar los siguientes subconjuntos de TxO asociados a el:

- $W_{UTxO} = \{u \in UTxO : \exists k \in W \mid \pi_0(u)(k) = true\}$, la suma de los valores de estos TxO forman el balance de dicho *monedero*.
- $W_{NoTxO} = \{o \in NoTxO : \exists k \in W \mid \pi_0(o)(k) = true\}$, de aquí se extraen nuevos $UTxOs$ que permiten al *monedero* reingresar el excedente producido por una transacción propia a su dominio.
- $W_{STxO} = \{s \in STxO : \exists k \in W \mid \pi_0(s)(k) = true\}$. Este conjunto conforma el historial de gastos del *monedero*.
- $ExtTxO = \overline{W_{TxO}} \cap TxO$. Aquí encontramos todos aquellos $TxOs$ que no pertenecen al *monedero* W, y que podrían conformar un $UTxO$.

Definición 9 (Descriptor de una transacción)**.** *Sea un monedero $W \subseteq Proof$, dado $I \subseteq W_{UTxO}, O \subseteq ExtTxO, C \subseteq W_{NoTxO}$ y $e \in \mathbb{N}_0$, llamaremos descriptor de una transacción a la 4-upla (I, O, C, e), tal que define las siguientes transformaciones:*

- $UTxO = (UTxO \setminus I) \cup O \cup C$, *en particular* $W_{UTxO} \cup C$
- $STxO = STxO \cup I$
- $NoTxO = NoTxO \setminus (C \cup O)$

La definición anterior nos permite establecer finalmente la estructura con la que describiremos las operaciones llevadas a cabo en una transacción en Bitcoin. Esta describe la transición de pertenencia de un TxO de una partición a otra, y respectivamente, la transferencia de valor realizada (fig. 5).

Definición 10 (Salida de cambio)**.** *Dado un monedero $W \subseteq Proof$, y dado un descriptor de transacción $T = (I, O, C, e)$ generado por W, entenderemos por salida de cambio a todo elemento perteneciente al conjunto C de dicho descriptor.*

Definición 11 (Entrada de una transacción)**.** *Dado un monedero $W \subseteq Proof$, y dado un descriptor de transacción $T = (I, O, C, e)$ generado por W, entenderemos por entrada de una transacción a todo elemento perteneciente al conjunto I de dicho descriptor.*

Definición 12 (Pago)**.** *Dada un monedero $W \subseteq Proof$, y dado un descriptor de transacción $T = (I, O, C, e)$ generado por W, entenderemos por Pago a todo elemento perteneciente al conjunto O de dicho descriptor.*

Definición 13 (Excedente)**.** *Dada una monedero $W \subseteq Proof$, y dado un descriptor de transacción $T = (I, O, C, e)$ generado por W, entenderemos por Excedente al valor e de*

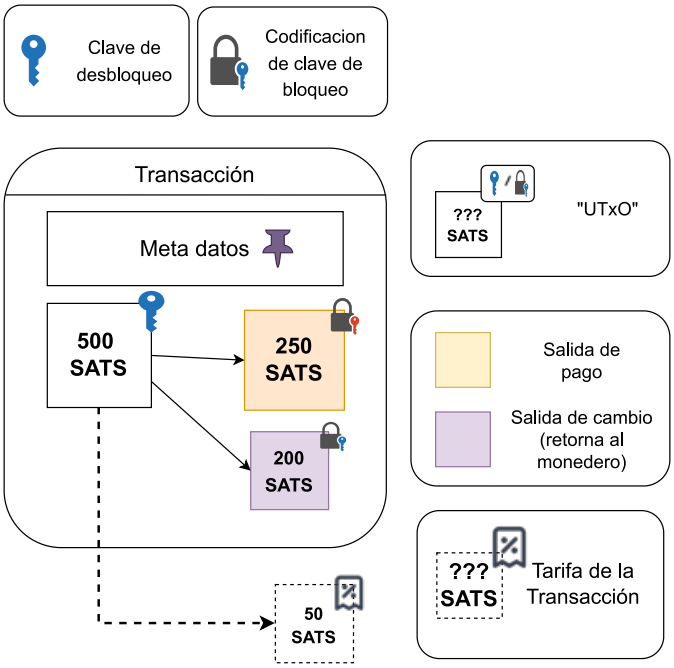


Figura 5: Esquema de una transacción

dicho descriptor.

No debemos olvidar que estas definiciones de carácter teórico tienen una correspondencia con estructuras reales del protocolo, que se almacenan y transmite en forma de datos en las distintas implementaciones del mismo. Como las transacciones compiten por el espacio dentro de cada bloque, se han ideado formas de calcularlo , que permiten incentivar las que realizan un uso mas eficiente del mismo. A partir de ahora, cuando hablemos de unidades de peso, nos estaremos refiriendo a estas unidades especiales utilizadas por el protocolo. Para representar dicha ponderación, asumiremos la existencia de dos funciones:

$$weight_{in} : Proof \times \{UTxO\} \to \mathbb{R}_0$$
$$weight_{out} : \{UTxO\} \to \mathbb{R}_0$$

Su implementación no es más que una suma ponderada de las partes del tipo de elemento de su dominio, por lo que se mantiene la linealidad de los modelos.

Para simplificar el cálculo del peso de un $UTxO$ perteneciente a un *monedero* W, utilizado como entrada de una transacción, emplearemos la siguiente notación:

Notación 2.

$$weight_{in}^{W} : \{W_{UTxO}\} \to \mathbb{R}_0$$
$$s.t. \quad weight_{in}^{W} = weight_{in}|_{W \times W_{UTxO}}$$

Definición 14 (Peso de una transacción)**.** *Dadas las funciones $weight_{in}$, $weight_{out}$, un monedero $W \subseteq Proof$ y el conjunto de los descriptores de transacciones T generados por dicho monedero, definiremos con:*

$$weight_W : T \to \mathbb{R}_0$$

A la función que establece un relación entre la transacción creada por un monedero y el valor ponderado final de la transacción subyacente, de acuerdo a la unidad definida por el protocolo.

Notación 3. *Utilizaremos las siguientes constantes:*

- *ρ: El peso de la porción de meta datos fija y común a toda transacción, expresado en las unidades definidas por el protocolo.*
- *γ: Una estimación de la tarifa por unidad de peso actual para poder incluir una transacción en la cadena en el número de bloques deseado por el usuario. En este trabajo no nos adentraremos en como estimar esta tarifa.*
- *$\gamma\prime$: Una tarifa por unidad de peso de referencia. Esta es una tarifa considerada estable de acuerdo a la demanda de espacio en la cadena de bloques.*

Definición 15 (Valor efectivo de un *UTxO*)**.** *El valor efectivo de una entrada o salida de una transacción es la cantidad de bitcoins bloqueada menos la tarifa total calculada para su inclusión en la transacción. Es decir, dado un monedero W tal que $u \in W_{UTxO}$ y una tarifa por unidad de peso γ, su valor efectivo es igual a:*

$$\pi_1(u) - weight_{in}^{W}(u) \cdot \gamma$$

Denotaremos con $ve(u)$ al valor efectivo de un UTxO tomando el monedero y la tarifa por unidad de peso del contexto.

Definición 16 (Desperdicio de una transacción)**.** *Dada una monedero $W \subseteq Proof$, un descriptor de transacción $Tx = (I, O, C, e)$, la tarifa por unidad de peso γ y la tarifa por*

unidad de peso de referencia $\gamma\prime$, definiremos:

$$\begin{aligned} Waste(W,T,\gamma,\gamma\prime) \coloneqq & \sum_{i\in I} weight^W_{in}(i)\cdot(\gamma-\gamma\prime) && \text{(Costo de preferencia temporal)}\\ & + \delta_{c,0}\cdot e && \text{(Costo de producir excedente)}\\ & + (1-\delta_{c,0})\cdot\sum_{c\in C}(weight_{out}(c)\cdot\gamma + weight^W_{in}(c)\cdot\gamma\prime) && \text{(Costo de producir cambio)} \end{aligned}$$

Denotaremos a esta función con el nombre $Waste$ y a su valor lo llamaremos el desperdicio de una transacción.

Definición 17 (Transacción Válida)**.** *Dado un monedero $W \subseteq Proof$, diremos que un descriptor de transacción $Tx = (I, O, C, e)$ tal que $I \subseteq W_{UTxO}, O \subseteq ExtTxO, C \subseteq W_{NoTxO}$ y $e \in \mathbb{N}_0$ es válido si:*

$$\begin{aligned} \sum_{u\in I}\pi_1(u) &\geq \sum_{o\in O}\pi_1(o) + weight_W(Tx)\cdot\gamma \\ \sum_{u\in I}\pi_1(u) - \sum_{k\in C}\pi_1(k) - e &= \sum_{o\in O}\pi_1(o) + weight_W(Tx)\cdot\gamma \end{aligned}$$

Definición 18 (Transacción libre de cambio)**.** *Dada una monedero $W \subseteq Proof$, diremos que un descriptor de transacción $Tx = (I, O, C, e)$ tal que $I \subseteq W_{UTxO}, O \subseteq ExtTxO, C \subseteq W_{NoTxO}$ y $e \in \mathbb{N}_0$ se considera **libre de cambio** si $|C| = 0$. Es decir, si el número de salidas de cambio es 0.*

Notación 4 (Umbral de donación)**.** *Utilizaremos la letra **h** para indicar el máximo que un monedero esta dispuesto a donar a los mineros al momento de realizar una transacción. Es decir, el límite a partir del cual se decide o no la creación de una salida de cambio para dicha transacción.*

Notación 5 (Umbral de transacción económicamente viable)**.** *Utilizaremos la letra **d**, para indicar el valor a partir del cual se considera que la salida de una transacción tiene un valor efectivo positivo en el caso de incluirla como entrada en una futura transacción a la tarifa por unidad de peso de referencia.*

Definición 19 (Transacción aceptable)**.** *La transacción sera aceptada en la cadena de bloques de Bitcoin si es válida y dado su descriptor $Tx = (I, O, C, e)$ se cumple uno de los siguientes casos.*

a) $|C| = 0$ *y* $0 \leq e \leq h$ *(el excedente no supera el límite para generar salidas de cambio)*

b) $e = 0$ *y* $\forall k \in C, k \geq d$ *(toda salida de cambio tiene un valor efectivo positivo)*

Cuando se cumplan estas condiciones, llamaremos transacción aceptable a la misma.

La definición anterior nos permite acotar el conjunto de las transacciones que tienen posibilidad de ser incluidas en la cadena de bloques. El protocolo y cada uno de los nodos acepta y asegura el cumplimiento de estas restricciones, entre otras.

Notación 6 (Plantilla de *UTxO* de cambio)**.** *Utilizaremos la letra* κ *para indicar la plantilla que utiliza el monedero para calcular los costos de crear una salida de cambio en una transacción. Para simplificar las simulaciones, fijaremos el tipo de la misma en P2WPKH.*

3.3 Problema

Dada una colección de solicitudes de pago $O \subseteq ExtTxO$, buscamos $I \subseteq W_{UTxO}$, $C \subseteq W_{NoTxO}$, $e \in \mathbb{N}_0$, de tal forma que $Tx = (I, O, C, e)$ es una transacción aceptable.

A partir de esta formulación general del problema veremos que se pueden encontrar múltiples soluciones, optimizadas para generar transacciones de determinada composición, pero siempre aceptables.

3.4 Modelos

A continuación presentamos tres modelos para generar transacciones aceptables de acuerdo a distintos criterios de optimización.

3.4.1 Avoid change

El primero de ellos esta fuertemente inspirado en el primer modelo utilizado por Diroff en el algoritmo *Coin Selection with Leverage*, y busca generar transacciones que no produzcan *UTxOs* de cambio, es decir, agrega todo remanente al excedente de la transacción.

Definimos este modelo mediante la ecuación:

$$\begin{aligned} \arg\min_{j} \quad & \sum_{j=1}^{|I|} x_j \cdot ve(u_j) - \sum_{p \in O} (\pi_1(p) + weight_{out}(p) \cdot \gamma) - \rho \cdot \gamma \\ \text{s.t.} \quad & W \subseteq Proof, \\ & Tx = (I, O, C, e), \\ & O \subseteq ExtTxO, I \subseteq W_{UTxO} \\ & e \in \mathbb{N}_0 \end{aligned} \tag{2}$$

Sujetas a las siguientes restricciones:

(1) Es una transacción válida:

$$\begin{aligned} \sum_{u \in I} \pi_1(u) & \geq \sum_{p \in O} \pi_1(p) + weight_W(Tx) \cdot \gamma \\ \sum_{u \in I} \pi_1(u) - \sum_{k \in C} \pi_1(k) - e & = \sum_{p \in O} \pi_1(p) + weight_W(Tx) \cdot \gamma \end{aligned}$$

(2) El excedente producido no supera al costo de producir una salida de cambio a la tarifa por unidad de peso actual:

$$\sum_{j=1}^{|I|} x_j \cdot ve(u_j) - \sum_{p \in O} (\pi_1(p) + weight_{out}(p) \cdot \gamma) - \rho \cdot \gamma \leq (weight_{in}^{W}(\kappa) + weight_{out}(\kappa)) \cdot \gamma$$

(3) El excedente producido es mayor o igual a cero:

$$\sum_{j=1}^{|I|} x_j \cdot ve(u_j) - \sum_{p \in O} (\pi_1(p) + weight_{out}(p) \cdot \gamma) - \rho \cdot \gamma \geq 0$$

(4) La transacción es libre de cambio: $C = \emptyset$

(5) Cada variable x_j es binaria:

$$x_j \in \{0, 1\} \quad \forall j \in \{1, ..., |I|\}$$

Notar que las restricciones 1, 2, 3 y 4 determinan que se trata de una transacción aceptable.

3.4.2 Minimize waste

El segundo modelo esta basado en la implementación actual del algoritmo de selección de monedas en Bitcoin Core y en la métrica de desperdicio surgida de este.

Este modelo esta definido por la ecuación:

$$\begin{aligned} \underset{I,C,e}{\arg\min} \quad & Waste(W, Tx, \gamma, \gamma\prime) \\ \text{s.t.} \quad & W \subseteq Proof, \\ & Tx = (I, O, C, e), \\ & O \subseteq ExtTxO, I \subseteq W_{UTxO}, C \subseteq W_{NoTxO}, \\ & e \in \mathbb{N}_0 \end{aligned} \tag{3}$$

Sujeta a las siguientes restricciones:

(1) Es una transacción aceptable:

$$\sum_{u \in I} \pi_1(u) \geq \sum_{p \in O} \pi_1(p) + weight_W(Tx) \cdot \gamma,$$

$$\sum_{u \in I} \pi_1(u) - \sum_{k \in C} \pi_1(k) - e = \sum_{p \in O} \pi_1(p) + weight_W(Tx) \cdot \gamma,$$

$$|C| = 0 \text{ y } 0 \leq e \leq h \text{ ó } e = 0 \text{ y } \forall k \in C, k \geq d.$$

(2) Cada variable x_j es binaria:

$$x_j \in \{0, 1\} \quad \forall j \in \{1, ..., |I|\}$$

3.4.3 Maximize effective value

Este modelo es una variación del modelo *Minimize Waste*, inspirado en el algoritmo *"Double Target"* propuesto por Mark Erhardt y el algoritmo *Random Improve* propuesto por Edsko de Vries, pero solo utilizada ante la obligación de generar salidas de cambio por no disponer de una solución libre de ellas. En ese caso se busca que el valor efectivo del cambio este dentro de una desviación estándar de la mediana de la distribución del valor de las solicitudes de pagos.

La idea es que ante situaciones en las que indefectiblemente debamos producir una salida de cambio, esta acarree un valor útil para utilizar a futuro en el *monedero*.

Este modelo esta definido por la ecuación:

$$\begin{aligned} \underset{I,C,e}{\arg\min} \quad & Waste(W, Tx, \gamma, \gamma\prime) \\ \text{s.t.} \quad & W \subseteq Proof, \\ & Tx = (I, O, C, e), \\ & O \subseteq ExtTxO, I \subseteq W_{UTxO}, C \subseteq W_{NoTxO}, \\ & e \in \mathbb{N}_0 \end{aligned} \tag{4}$$

Sujeta a las siguientes restricciones:

(1) Es una transacción aceptable:

$$\sum_{u \in I} \pi_1(u) \geq \sum_{p \in O} \pi_1(p) + weight_W(Tx) \cdot \gamma,$$
$$\sum_{u \in I} \pi_1(u) - \sum_{k \in C} \pi_1(k) - e = \sum_{p \in O} \pi_1(p) + weight_W(Tx) \cdot \gamma,$$
$$|C| = 0 \text{ y } 0 \leq e \leq h \text{ ó } e = 0 \text{ y } \forall k \in C, k \geq d.$$

(2) En el caso de que haya salidas de cambio, el valor efectivo de cada uno de ellas esta dentro de una desviación estándar de la mediana de la distribución del valor de las solicitudes de pagos.

$$\forall k \in C: \quad \tilde{O} - \sigma^2(O) \leq ve(k) \leq \tilde{O} + \sigma^2(O)$$

donde $\tilde{O}$ es la mediana del conjunto O.

(3) Cada variable x_j es binaria:

$$x_j \in \{0, 1\} \quad \forall j \in \{1, ..., |I|\}$$

4 Experimentación

Para la etapa de simulación se programó una aplicación ejecutable por línea de comandos en Python que permite la lectura de escenarios de simulación a partir de archivos y produce resultados en formato *csv* para ser interpretados de forma libre. Su implementación es de código abierto y esta disponible en un repositorio de acceso público bajo el dominio `https://github.com/csralvall/tx-optimizer`.

Dado que nuestro objetivo es realizar un análisis únicamente del proceso de selección de monedas, no realizaremos pruebas en ninguna red real del protocolo debido a los datos superfluos e imprecisiones que esto generaría. Por lo tanto, el sistema implementado utiliza los componentes mínimos para poder simular el comportamiento de un sistema de selección de monedas de un monedero de Bitcoin real. Incluso algunos elementos no son implementados en toda su complejidad debido a los límites de este trabajo, pero su código es lo suficientemente general como para permitir la extensión de los mismos en caso de interés.

Los archivos admitidos tiene un formato de lista de triplas que contienen un identificador del bloque de pertenencia, el monto de la transacción y la tarifa por unidad de peso al realizarla. Si la cantidad es positiva se trata de un ingreso, si es negativa es una obligación de pago. El identificador de bloque permite crear pagos múltiples, aunque no ha sido utilizado para simular en el presente trabajo. El formato de las transacciones es de n entradas y una o más salidas, incluyendo la salida opcional de cambio y los pagos que se necesiten producir. En general se produce un solo pago, pero en caso de que selecciones pasadas hayan fallado se incluyen los pagos fallidos en la siguiente selección. El tipo de *UTxO* ha sido fijado en P2WPKH, por simplicidad. No se utiliza el seguimiento de la permanencia de un *UTxO* en el monedero, ni tampoco se admite la existencia de dos *UTxOs* asociados a la misma dirección, como puede suceder cuando se reutilizan direcciones de Bitcoin. Esto podría modificarse a futuro.

4.1 Datos capturados

Cuando el proceso de selección para un pago finaliza de forma exitosa o no, se registran los datos en formato *csv* para su posterior análisis. Por cada transacción, en dicho documento se almacenan los siguientes datos:

- identificador de la transacción
- nombre del algoritmo o indicador de fallo de acuerdo al caso
- balance del monedero al momento de realizar la transacción
- cantidad de *UTxOs* luego de realizar la transacción en el monedero
- cantidad de salidas de pago
- cantidad de salidas de cambio
- cantidad de entradas con valor efectivo negativo

- cantidad de satoshis donados a los mineros
- cantidad de satoshis recuperados como salidas de cambio
- desperdicio de la transacción de acuerdo a la métrica *Waste*
- tarifa total incurrida por la transacción
- tarifa por unidad de peso final de la transacción
- tarifa por unidad de peso objetivo
- tiempo requerido de procesamiento en CPU (sin incluir otros procesos que se puedan estar ejecutando en simultáneo)

4.2 Modelos implementados

Se implementaron cinco modelos en total para correr las simulación, tres de los cuales fueron modelados en el capítulo anterior. Los dos restantes, debido a su simplicidad, no fueron modelados bajo la óptica de la programación binaria lineal, pero serán utilizados para demostrar otras formas de aproximación a la solución, una *greedy* y otra *aleatoria*.

- *greatest-first*: Algoritmo *greedy* que selecciona los *UTxOs* con mayor valor efectivo hasta cubrir los requerimientos de los pagos. Implementado como algoritmo testigo que permita comparar los algoritmos más avanzados con estrategias básicas.
- *single-random-draw*: Otro algoritmo testigo que selecciona de forma aleatoria los *UTxOs* de acuerdo a su valor efectivo hasta que cubre todos los pagos requeridos. Este ha sido tomado como algoritmo base en algunos casos de análisis, debido a su capacidad de representar la distribución de los *UTxOs* en el monedero.
- *avoid-change*: Este algoritmo, definido formalmente mediante la ecuación (2) y sus restricciones, busca generar transacciones evitando las salidas de cambio, es decir, liberando cualquier excedente como donación a los mineros.
- *minimize-waste*: Basado en la métrica *Waste*. La formulación teórica del modelo, con la función objetivo definida por la ecuación (3) y las restricciones asociadas, no puede ser trasladada directamente al código debido a la existencia de casos distintos y disjuntos de acuerdo a si se crean salidas de cambio o se libera el excedente como donación a los mineros. Para ello se utilizan dos modelos distintos, *minimize-waste-without-change* y *minimize-waste-with-change*. Seleccionando la solución que minimiza la métrica *Waste*.
- *maximize-effective-value*: Definido en la ecuación (4) junto con sus restricciones, este es otro algoritmo ensamblado. Comparte con *minimize-waste* el modelo *minimize-waste-without-change* para generar transacciones sin cambio. El modelo para generar transacciones con cambio es *aim-payment-amount-as-change*, el cual busca acercar la cantidad de bitcoins asignada a la salida de cambio a un número dentro de una desviación estándar de la mediana de los valores de las solicitudes de pago, a diferencia del modelo *minimize-waste-with-change*, que genera el mínimo cambio posible.

4.3 Resultados de la simulación

Para correr las simulaciones se utilizaron los siguientes escenarios, obtenidos del simulador de selección de monedas implementado en Python por Andrew Chow y Mark Erhardt[17]:

- Variantes de *bustabit-2019-2020*: Estos escenarios fueron generados a partir de un corpus de depósitos y extracciones de aproximadamente 540 mil transacciones, realizadas en un servicio de apuestas online basado en Bitcoin llamado *Bustabit*[16], combinados con un total de 730 estimaciones diarias de la tarifa por unidad de peso para colocar una transacción en los siguientes 6 bloques, extraídas del servicio *txstats*[17]. Estas estimaciones siguen un patrón ascendente con dos picos intercalados por una zona de baja variación. La relación entre depósitos y extracciones es de 2 a 1 aproximadamente. Debido a la diferencia en el tamaño de la muestra de transacciones y el de la muestra de estimaciones de la tarifa por unidad de peso, el patrón tarifario se repite en la secuencia la cantidad de veces que sea necesario.
- *random_blocks*: Escenario generado a partir del procesamiento aleatorio de bloques de la cadena, mediante la consulta a un nodo particular.

Este trabajo utilizó recursos computacionales compartidos por la FAMaF y el CCAD de la Universidad Nacional de Córdoba[18], que forman parte del SNCAD del MinCyT de la República Argentina.

Aunque la resolución de los modelos utilizaba paralelismo a nivel de threads provisto por *Coin-OR/CBC*, las simulaciones se corrieron de forma secuencial sobre las distintas combinaciones de escenarios y modelos con un tiempo de utilización del CPU rondando las 45hs.

Para facilitar la visualización de los resultados se generaron distintas tablas y gráficos.

4.3.1 bustabit-2019-2020 y derivados

El escenario *bustabit-2019-2020* completo, con 548981 transacciones, es el mas grande de todos los evaluados. La relación entre ingresos y pagos es de aproximadamente 2 a 1 (fig. 6).

El algoritmo que permitió la mejor conservación del balance del monedero fue *greatest-first*, dejando en segundo lugar a *minimize-waste* y en tercero a *maximize-effective-value*, aunque la diferencia relativa no es significativa (tab. 1, fig. 8).

maximize-effective-value, a pesar de la diferencia de balance con *minimize-waste*, mejora el promedio del tamaño del monedero en aproximadamente un 30%, lo cual es de importancia si consideramos que al momento de resolver los modelos, es una reducción de igual magnitud en el espacio de soluciones.

[16] `https://www.bustabit.com/`
[17] `https://txstats.com/`
[18] `https://ccad.unc.edu.ar/`

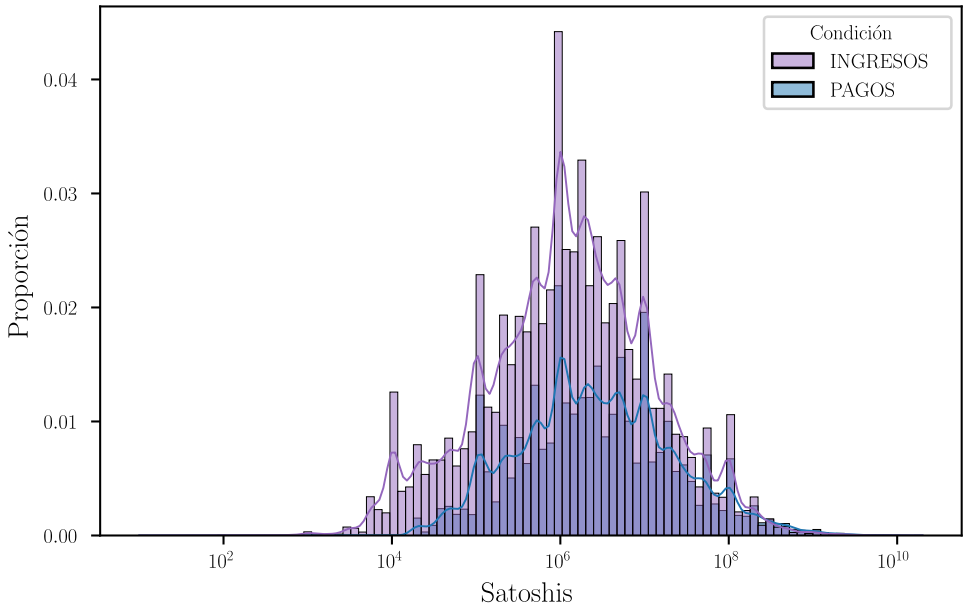


Figura 6: Distribución entradas y solicitudes de pago, *bustabit-2019-2020*

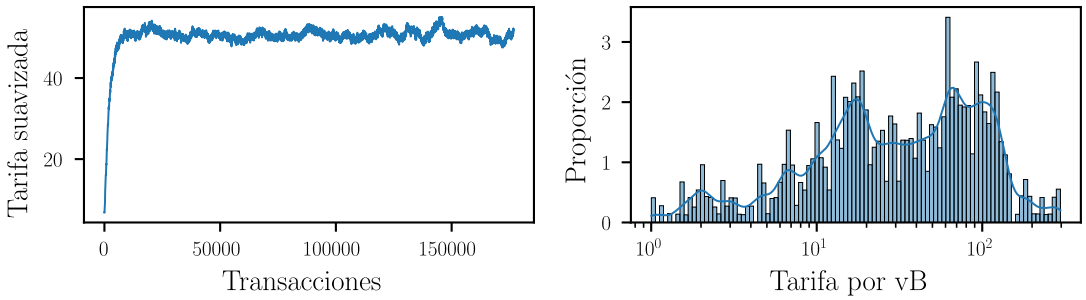


Figura 7: Distribución y evolución de la tarifa por unidad de peso, *bustabit-2019-2020*

modelo	**balance final** ($x10^{11}$ **SATS**)	**#monedero máximo**	**mediana #monedero**	**máximo #entradas**	**participación**
greatest first	2.6914	258253	145261	1929	100.00%
single random draw	2.6828	7892	3409	653	100.00%
maximize effective value	2.6872	1680	1039	243	100.00%
minimize waste	2.6880	2245	1259	255	100.00%
avoid change	2.6845	571	194	167	31.05%

Tabla 1: Resumen de actividad del monedero, *bustabit-2019-2020*

1^{er} mejor 2^{do} mejor 1^{er} peor

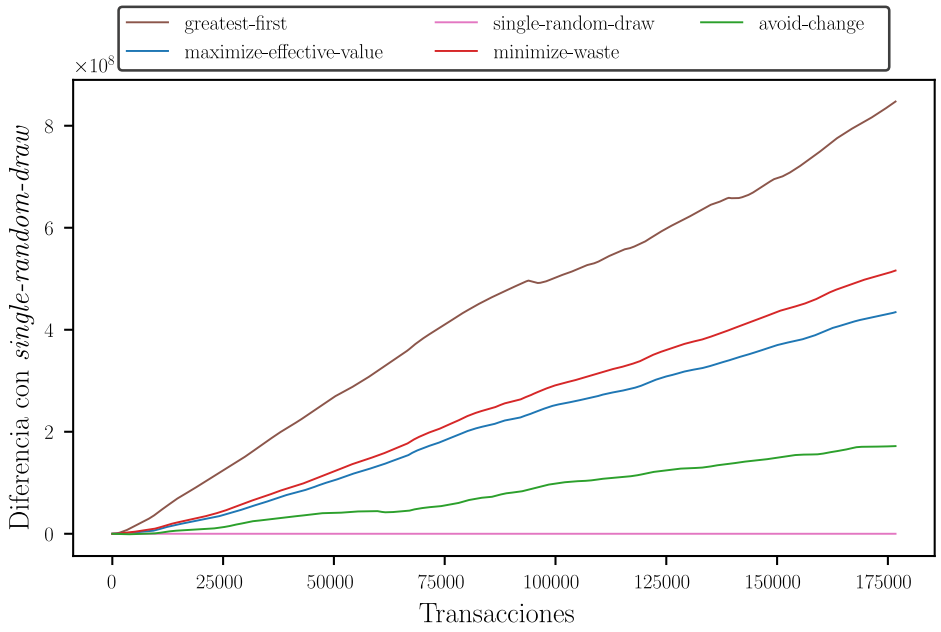


Figura 8: Balance comparado con *single-random-draw*, *bustabit-2019-2020*

greatest-first al contrario, expone en este caso la gran debilidad de esta aproximación *greedy* a la solución: el no utilizar más que los *UTxOs* de valor máximo para la selección, genera cambio de forma constante, y el no manejo del mismo se traduce en un crecimiento exponencial en el tamaño del conjunto de *UTxOs* bajo el dominio del monedero (fig. 9).

Esto impacta de forma negativa en distintos niveles:

1. Genera mayor demanda de recursos por parte del monedero, puesto que mantener el conjunto de *UTxOs* en una memoria de acceso rápido se encarece y se requiere más tiempo para recorrer el conjunto completo.
2. La demanda en el monedero afecta al resto de los nodos del protocolo, dado que la contabilidad de *UTxOs* en existencia esta distribuida en todos ellos. Si consideramos los efectos de una aplicación masiva de este algoritmo, el efecto multiplicador sería devastador para cada nodo en particular, aumentando los requerimientos de hardware, elevando los costos, quitando participantes y reduciendo la descentralización.
3. La generación de cambio de forma automática sin considerar el *valor efectivo* de los mismos, le quita flexibilidad al monedero para adaptarse a los distintos escenarios, lo cual puede dejarlo inutilizable en escenarios donde las tarifas aumentan considerablemente.

Otro aspecto a observar es la poca participación de *avoid-change* en la generación

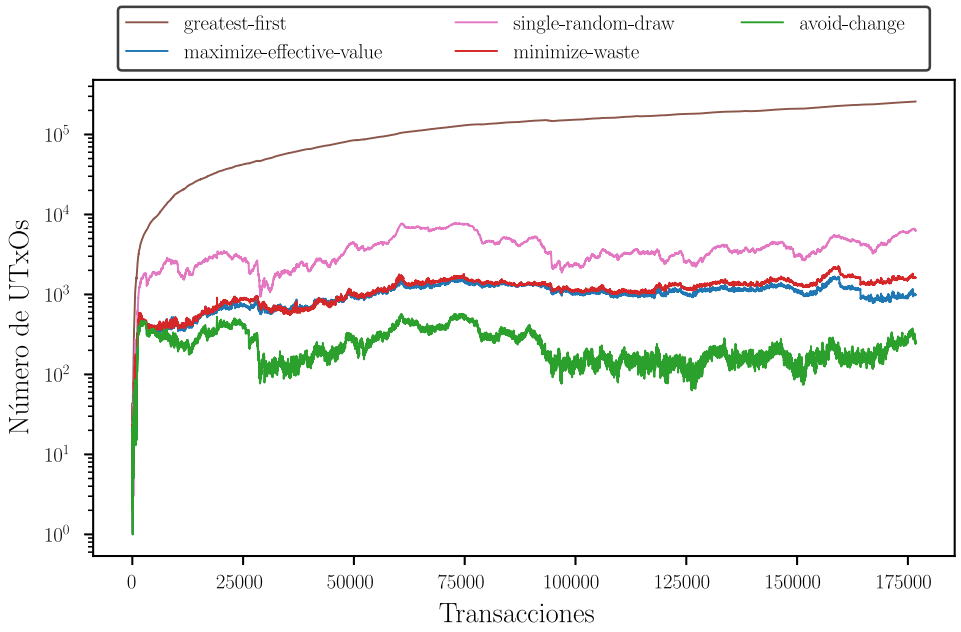


Figura 9: Tamaño del monedero, *bustabit-2019-2020*

de soluciones. Cerca del 70% de las selecciones utilizando dicha política fueron resueltas por el algoritmo de emergencia, *single-random-draw* (tab. 1). Es decir, la probabilidad de no encontrar una solución con *avoid-change* es muy alta. Con este modelo, se dan situaciones en las que la existencia de una solución sin cambio sucede a costa de donar un excedente generoso a los mineros, elevando efectivamente la tarifa por unidad de peso de la transacción, (fig. 15). Si consideramos el excedente de una transacción como un descuento al balance conservado por el monedero que genera la transacción, el efecto acumulado se hace evidente (fig. 8).

Esta utilización recurrente del algoritmo por defecto en el caso de *avoid-change* hace que esta política tenga un comportamiento muy influenciado por *single-random-draw* para algunas mediciones. Tal es el caso de la cantidad de *UTxOs* disponibles para el monedero, (fig. 9) donde la gráfica de ambas tiene un perfil similar.

En general, no existen grandes diferencias que permitan destacar a *maximize-effective-value* como una alternativa real a *minimize-waste*. Si las observamos con atención, la tendencia a buscar un cambio de mayor denominación de *maximize-effective-value* genera una reducción en los excedentes producidos y por ende, en la tarifa por unidad de peso (fig. 10). Pero, a su vez implica la inclusión de mayor número de entradas en la transacción para satisfacer el volumen deseado (fig. 12), encareciendo la tarifa total de la transacción (fig. 11). Esta tarifa ligeramente mayor debido a las entradas extras para *maximize-effective-value* es lo que lo aleja del balance final obtenido por *minimize-waste*

(fig. 13).

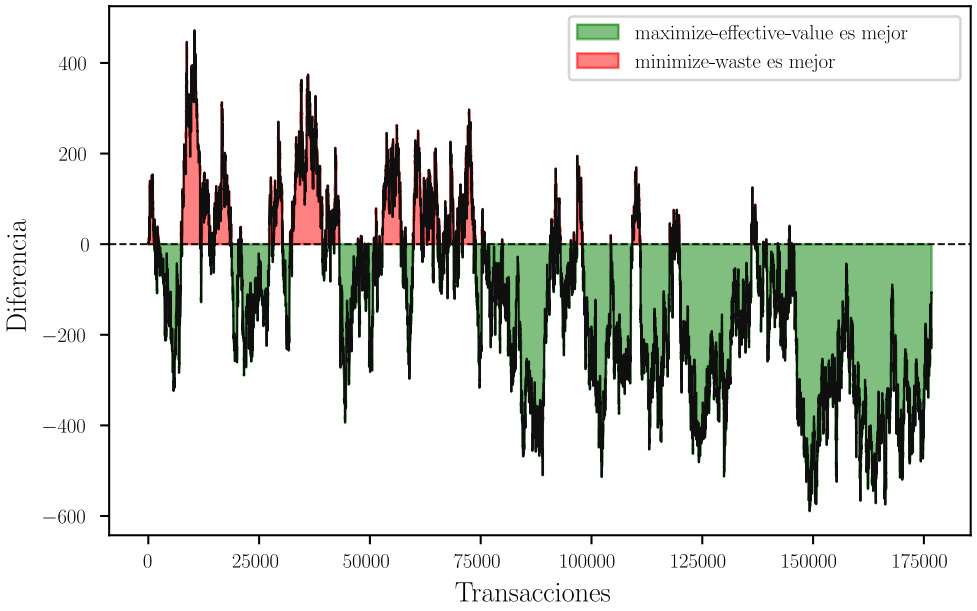


Figura 10: Diferencia entre tarifa objetivo y tarifa final por unidad de peso entre *maximize-effective-value* y *minimize-waste*, *bustabit-2019-2020*

En relación con las restantes políticas de selección, *maximize-effective-value* y *minimize-waste* son las que presentan una mayor previsibilidad al momento de pagar tarifas (fig. 15). Esto es importante para adaptarse a situaciones de gran variabilidad tarifaria, lo que coloca a ambos algoritmos, en los criterios de selección con mayor control sobre la tarifa por transacción.

Si contrastamos estos resultados con la cantidad de entradas por transacción, (fig. 16) podemos observar la clara relación entre la previsibilidad de las tarifas y el número de entradas. *greatest-first* es el algoritmo de selección que mas sufre esta asociación, debido al particionamiento continuo que hace del conjunto de *UTxOs*, que se traduce en la escasez de *UTxOs* idóneos para financiar transacciones en escenarios de tarifas altas, arrastrando un gran número de entradas en la selección y elevando la tarifa final de la transacción bajo esos casos. Las gráficas de *single-random-draw* siguen una lógica parecida. *avoid-change* en cambio, tiene mayor control del número de entradas y por ende, de la tarifa final, a costo de elevar la tarifa por unidad de peso produciendo mayores excedentes. Eso hace que al menos en lo que tarifas respecta, la similitud que señalábamos antes con *single-random-draw* (fig. 9), no se cumpla.

La distribución del cambio es la deseada en el caso de *maximize-effective-value*(fig. 17b), siguiendo una forma muy similar a la de los pagos (fig. 6). Al contrario, *minimize-waste*, tiene una distribución sesgada hacia la izquierda (fig. 17a), es decir, a los ordenes de

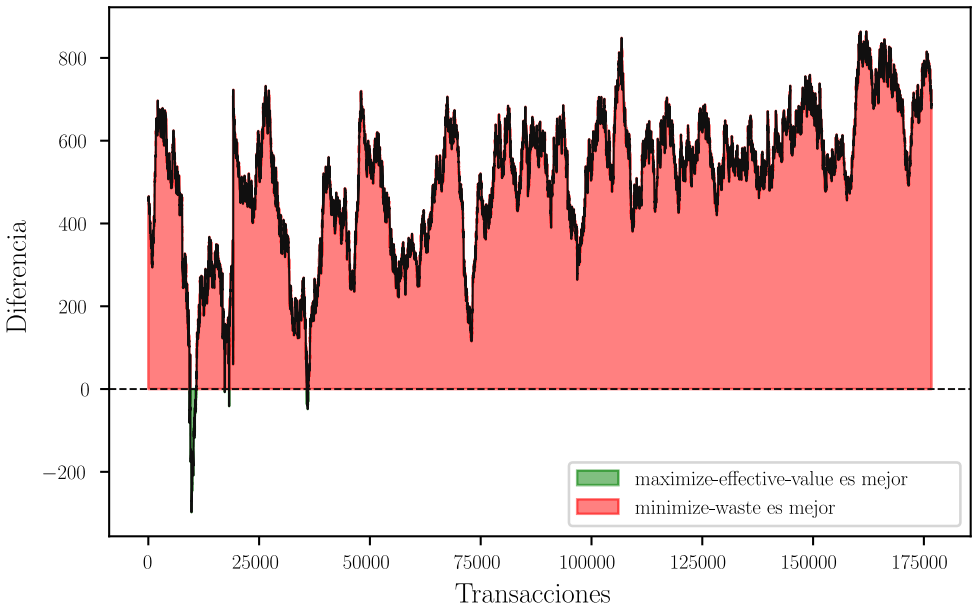


Figura 11: Diferencia de tarifa entre *maximize-effective-value* y *minimize-waste*, *bustabit-2019-2020*

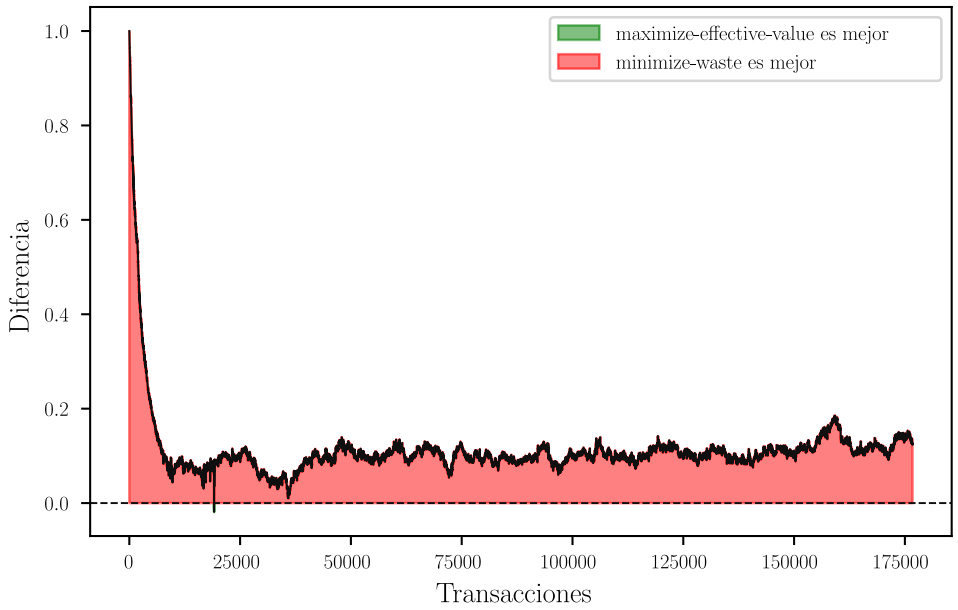


Figura 12: Diferencia de entradas entre *maximize-effective-value* y *minimize-waste*, *bustabit-2019-2020*

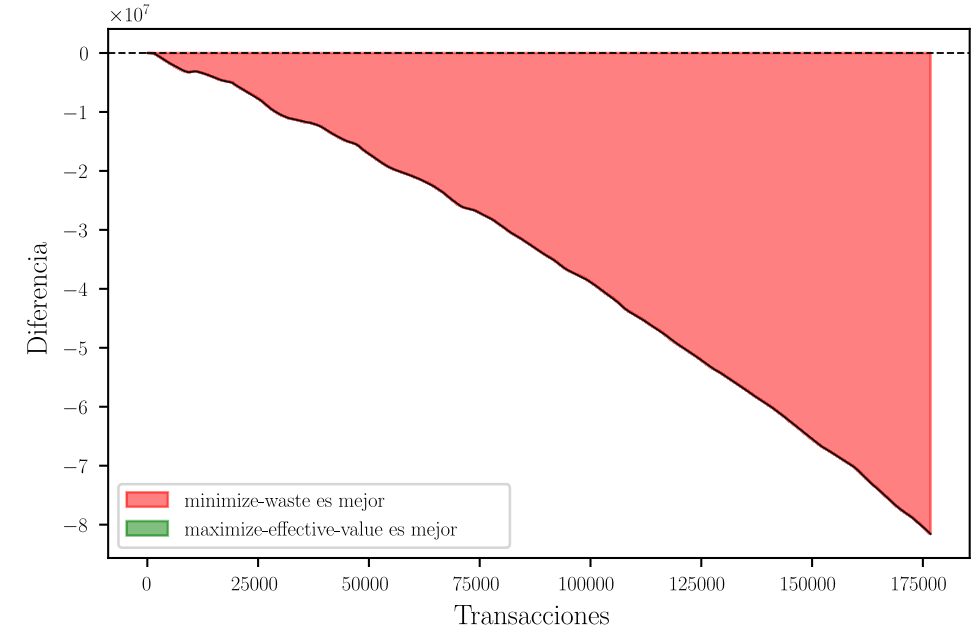


Figura 13: Diferencia de balance entre *maximize-effective-value* y *minimize-waste*, *bustabit-2019-2020*

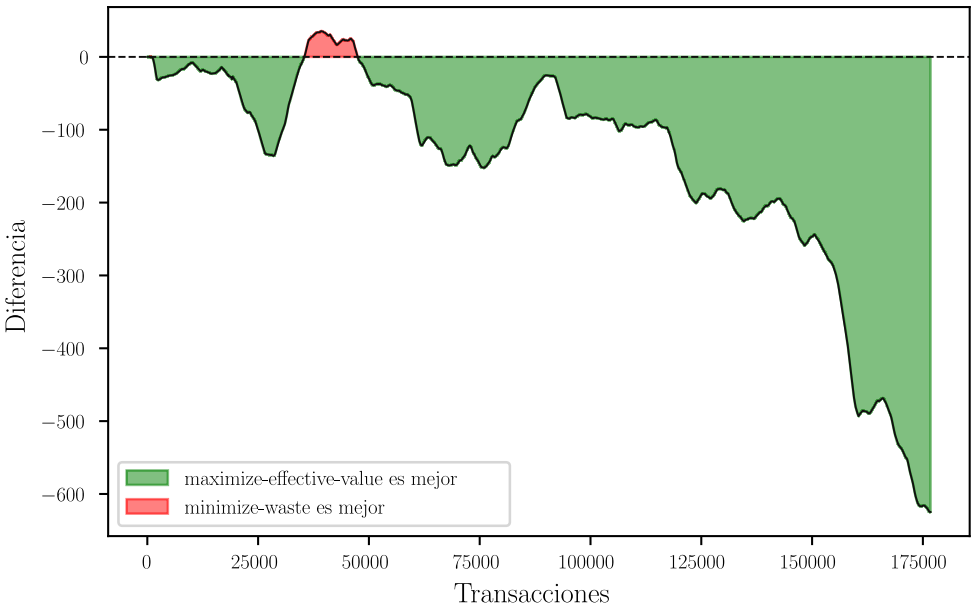


Figura 14: Diferencia entre el tamaño del monedero entre *maximize-effective-value* y *minimize-waste*, *bustabit-2019-2020*

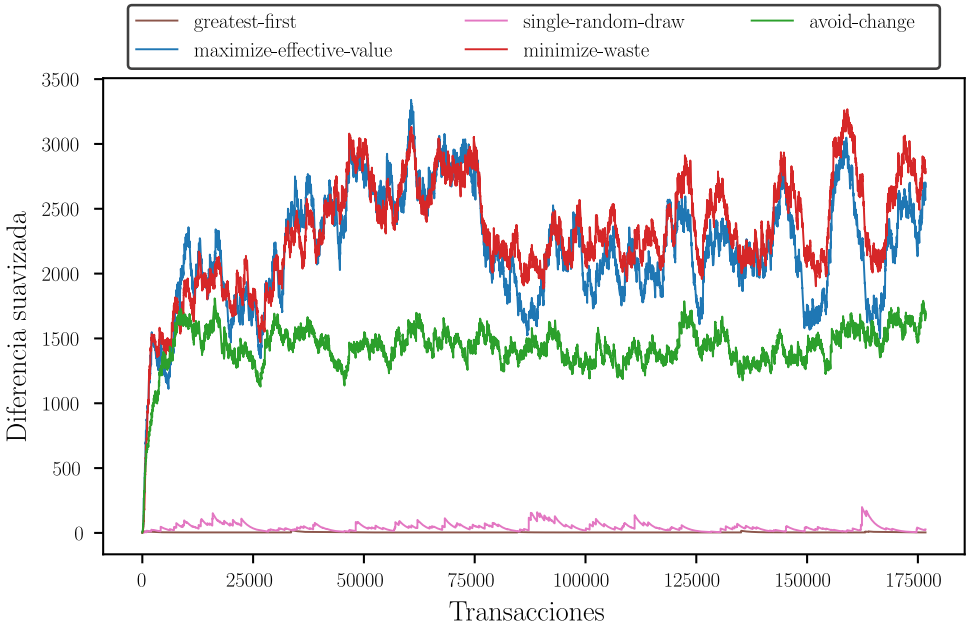


Figura 15: Diferencia entre tarifa objetivo y tarifa final por unidad de peso, *bustabit-2019-2020*

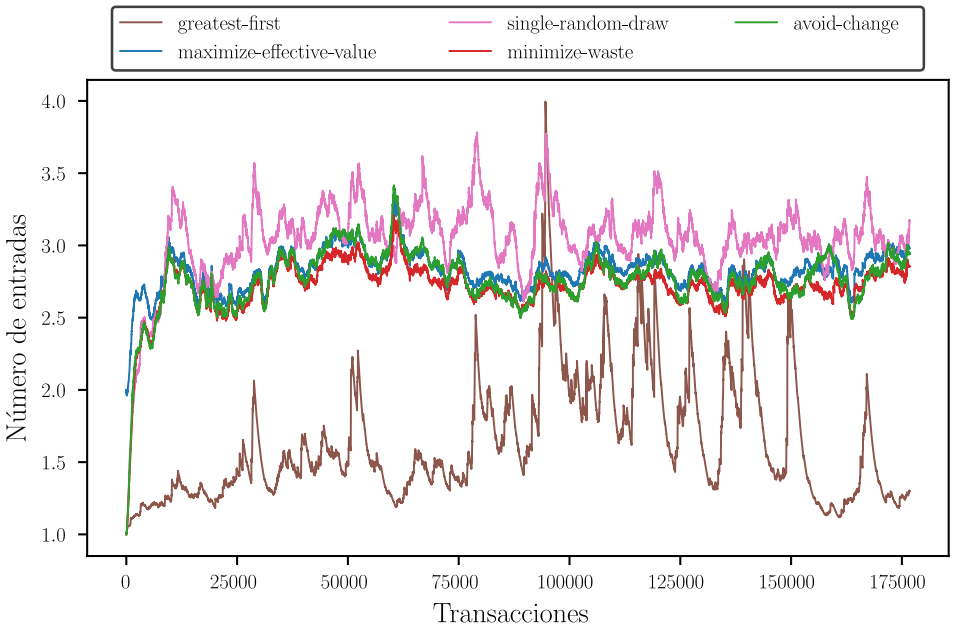
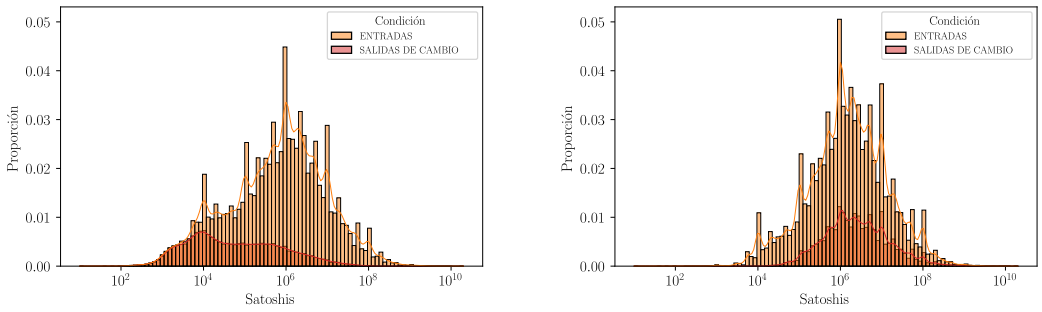


Figura 16: Cantidad de entradas por transacción, *bustabit-2019-2020*



(a) *minimize-waste* (b) *maximize-effective-value*

Figura 17: Distribución de entradas y salidas de cambio, *bustabit-2019-2020*

magnitud más pequeños. La razón de esto tiene que ver con que incluso en las situaciones en las que esta obligado a generar cambio, busca minimizarlo, generando salidas de cambio de denominación más pequeña que la de los pagos.

greatest-first se encuentra en el otro extremo (fig. 18), a razón de utilizar los *UTxOs* de mayor valor, también genera los cambios con mayor denominación, sesgando su distribución hacia la derecha, y aumentando la relación con respecto a las entradas utilizadas en las transacciones.

La proporción entre la distribución de entradas y salidas de cambio para *single-random-draw* esta sesgada hacia la derecha (fig. 19a), de manera semejante que la distribución de pagos. La distancia entre las medias de ambas distribuciones puede influir a que esta política tienda a incluir una entrada de las denominaciones mayores o iguales a la media de los pagos, en la búsqueda de superar el monto de los mismos junto con los costos de las tarifas.

avoid-change, acentúa aún más este sesgo (fig. 19b), debido a que siempre se busca primero evitar el cambio, utilizando todas las salidas de menor denominación en el proceso. Los fallos se producen cuando no es posible encontrar soluciones que no excedan el límite del umbral de donación. Dichos casos están asociados con pagos por valores elevados que no permiten ser alcanzados con *UTxOs* de denominación pequeña, causando que *single-random-draw* entre en acción y genere soluciones con cambio, utilizando mayoritariamente entradas de denominación alta debido al efecto de limpieza que tiene *avoid-change* sobre los *UTxOs* de menor valor. Bajo esta situación, se repite lo mencionado en el párrafo anterior, y *single-random-draw* vuelve a generar cambio por valores grandes, acentuado por el efecto de *avoid-change*.

Podemos observar estos efectos en la diferencia del valor efectivo preservado por *avoid-change* respecto de *single-random-draw* (tab. 2).

Si comparamos solamente los modelos diseñados para hacer un uso menos intensivo de las salidas de cambio (tab. 3), *minimize-waste*, *maximize-effective-value* y *avoid-change*, podemos observar que para *bustabit-2019-2020*, *minimize-waste* es el que genera

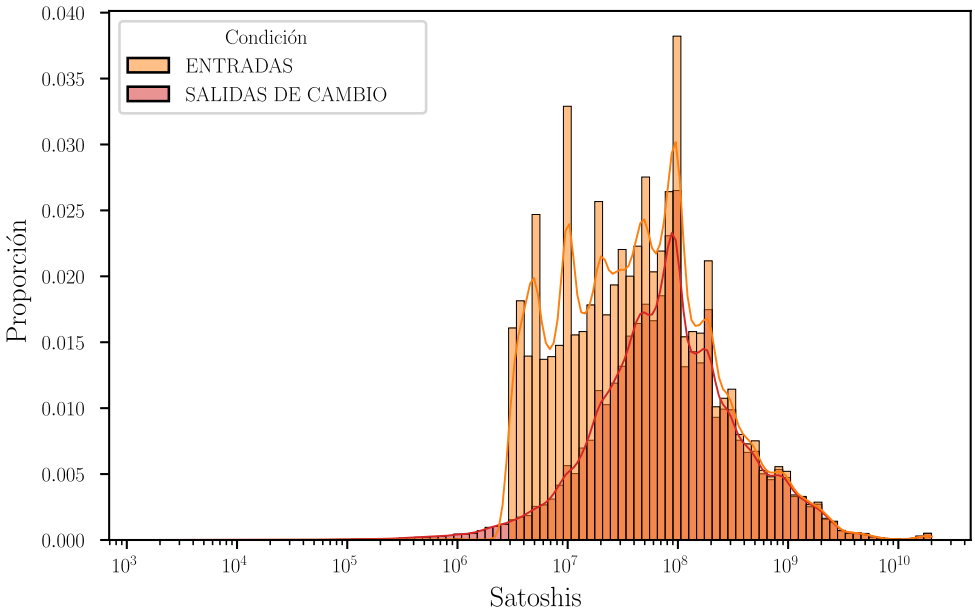
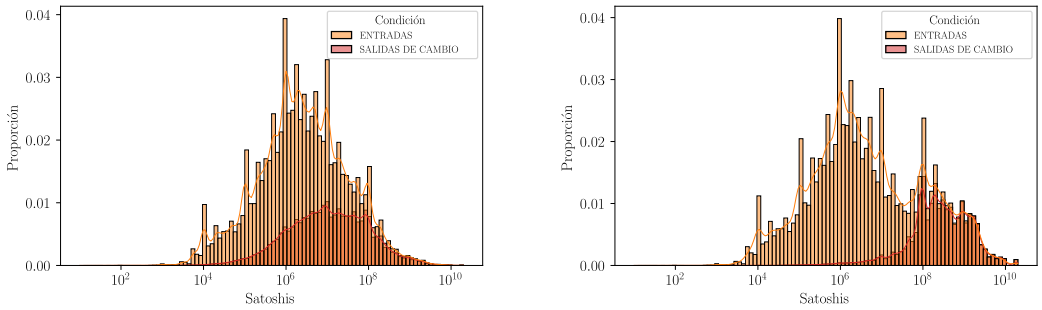


Figura 18: Distribución de entradas y salidas de cambio bajo el algoritmo *greatest-first*, *bustabit-2019-2020*



(a) *single-random-draw* (b) *avoid-change*

Figura 19: Distribución de entradas y salidas de cambio, *bustabit-2019-2020*

modelo	cantidad media de entradas	tarifa media por transacción	media preservada de valor efectivo
greatest first	-46.41%	-33.61%	174.65%
maximize effective value	-7.49%	-20.08%	-88.51%
minimize waste	-10.78%	-23.60%	-99.22%
avoid change	-8.96%	-9.26%	474.33%

Tabla 2: Desglose de transacción comparado a la selección aleatoria, *bustabit-2019-2020*

modelo	mediana de exceso	transacciones sin cambio
maximize effective value	684	26.11%
minimize waste	487	35.87%
avoid change	173	31.04%

Tabla 3: Comparación de manejo del excedente de modelos que evitan salidas de cambio, *bustabit-2019-2020*

el mayor porcentaje de transacciones sin salida de cambio en relación al total de ellas. El número de transacciones sin cambio para *avoid-change* ronda el 31%. Recordemos que este modelo solo producía alrededor de un 30% de las transacciones en su simulación, y que el 70% restante se debía a *single-random-draw* (tab. 1).

maximize-effective-value, a pesar de que la idea original de su diseño buscaba mejorar el desempeño del modelo en estos casos, queda en tercer lugar, además de con el mayor promedio de excedente producido por transacción, dándonos a entender, si contrastamos con el promedio de entradas (tab. 2), que las entradas extras (fig. 12), solo están destinadas a alcanzar los objetivos del valor de las salidas de cambio del modelo, pero no se traducen en la creación de *UTxOs* con valores más óptimos para el perfil de gastos del monedero.

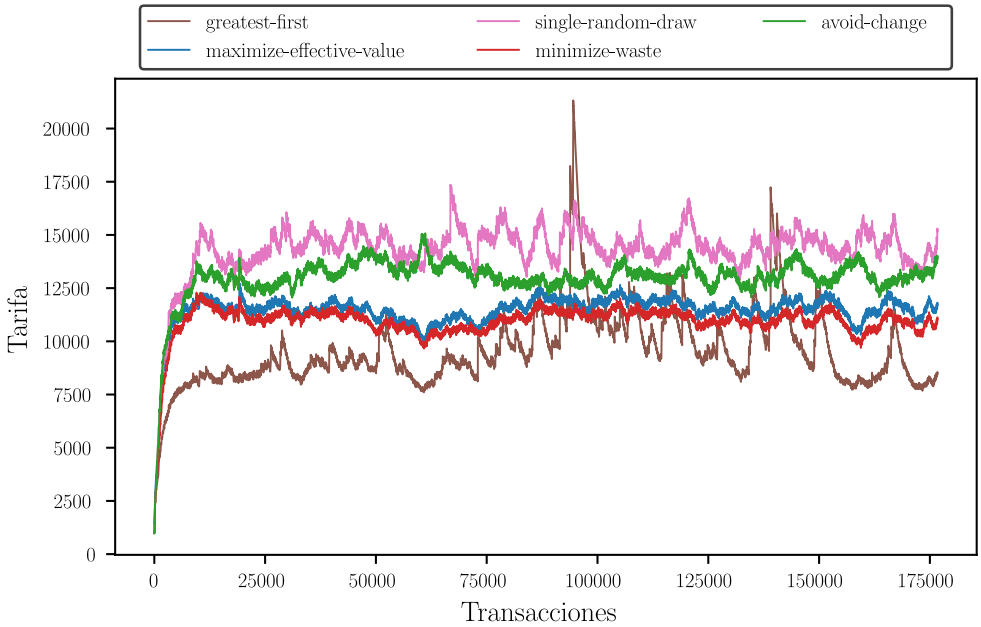


Figura 20: Tarifas por transacción, *bustabit-2019-2020*

No nos detendremos demasiado en los restantes escenarios derivados de *Bustabit*, ya

que al haber sido extraídos del mismo conjunto de datos, comparten muchas similitudes, como podemos observar en los resúmenes de actividad de los monederos durante la simulación.

La política de selección que mejor desempeño tiene en cuanto al balance preservado es *greatest-first* (Tablas 4 a 6), aunque su mala gestión del número de *UTxOs* la expone a los mismos tipos de variaciones observados en la gráfica del número de entradas por transacción o tarifas del modelo (Figuras 16 y 20).

minimize-waste mantiene su segunda posición en cuanto al balance. *maximize-effective-value*, en contraste, mejora el número de *UTxOs*, pero no logra un salto de magnitud respecto a *minimize-waste*. Esto junto a su peor desempeño en cuanto a la conservación del balance desestiman un remplazo de *minimize-waste* por parte de *maximize-effective-value*. En cuanto a *single-random-draw* y *avoid-change* podemos observar el efecto moderador que tiene el segundo sobre el primero al momento de la gestión de los *UTxOs*, en relación con su participación en la selección, a pesar de que los balances siempre dejan relegado a *avoid-change* al último lugar de la tabla de posiciones debido a su generosidad al momento de gestionar el exceso de las transacciones.

modelo	**balance final** ($x10^{10}$ **SATS**)	**#monedero máximo**	**mediana #monedero**	**máximo #entradas**	**participación**
greatest first	5.8351	29039	14510	69	100.00%
single random draw	5.8261	2919	1902	140	100.00%
maximize effective value	5.8287	704	406	108	99.95%
minimize waste	5.8292	711	418	92	100.00%
avoid change	5.8269	469	274	50	30.76%

Tabla 4: Resumen de actividad del monedero, *bustabit-2019-2020-short*

modelo	**balance final** ($x10^{10}$ **SATS**)	**#monedero máximo**	**mediana #monedero**	**máximo #entradas**	**participación**
greatest first	3.8574	8953	5315	48	100.00%
single random draw	3.8551	1935	1511	102	100.00%
maximize effective value	3.8554	505	382	97	99.84%
minimize waste	3.8556	592	400	82	100.00%
avoid change	3.8550	469	331	27	29.68%

Tabla 5: Resumen de actividad del monedero, *bustabit-2019-2020-tiny*

4.3.2 random-blocks

La principal diferencia de este escenario con respecto a las variaciones de *bustabit-2019-2020* es la mayor desviación que presenta la denominación de los *UTxOs* tanto para los ingresos como para los pagos, con un valor que ronda los $1e7$ satoshis. Ademas,

modelo	balance final ($x10^8$ SATS)	#monedero máximo	mediana #monedero	máximo #entradas	participación
greatest first	1.4581	4441	2644	581	100.00%
single random draw	1.4498	285	86	86	100.00%
maximize effective value	1.4538	132	19	92	99.92%
minimize waste	1.4560	143	25	91	100.00%
avoid change	1.4079	85	29	38	24.56%

Tabla 6: Resumen de actividad del monedero, *bustabit-2019-2020-tiny-hot-wallet*

presenta dos situaciones extremas en su distribución, una gran concentración de *UTxOs* con un valor que ronda los $6e3$ y los $1e7$ satoshis, tanto para pagos como para ingresos. En estos dos valores extremos de la distribución se acumulan cerca del 8% de los *UTxOs* de ingreso y el 4% de los de pago. Un porcentaje significativo si tenemos en cuenta que la media para los ingresos supera por poco margen el 2.5% y en el caso de los pagos el 1% de la proporción.

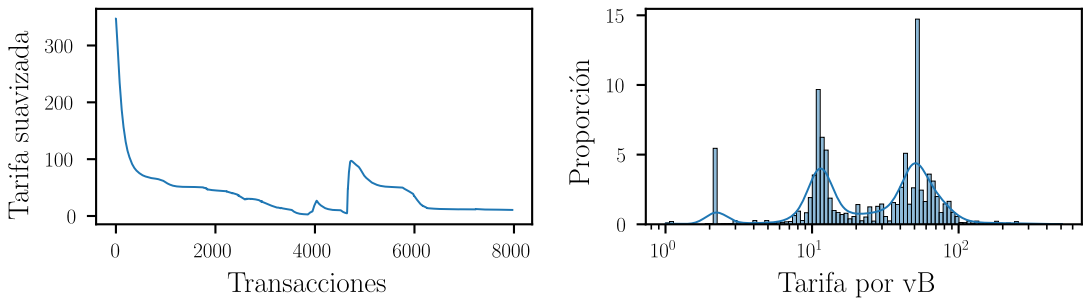


Figura 21: Distribución y evolución de la tarifa por unidad de peso, *random-blocks*

En cuanto al comportamiento de la tarifa por hora (fig. 21), presenta un patrón con una tendencia descendente general, interrumpida dos picos repentinos, espaciados entre sí.

En este caso los modelos dominantes en cuanto al control del conjunto de *UTxOs* (tab. 7), fueron *single-random-draw* y *avoid-change*, este último claramente debido al intensivo uso del primero, ya que su participación real fue levemente superior al 4%. Igualmente, se mantiene *greatest-first* primero en el volumen de balances, seguido de cerca por *minimize-waste*, aunque el escaso control del conjunto de *UTxOs* por parte del primero hace que el balance tenga variaciones violentas, colocándolo en circunstancias por debajo de los demás modelos (fig. 22).

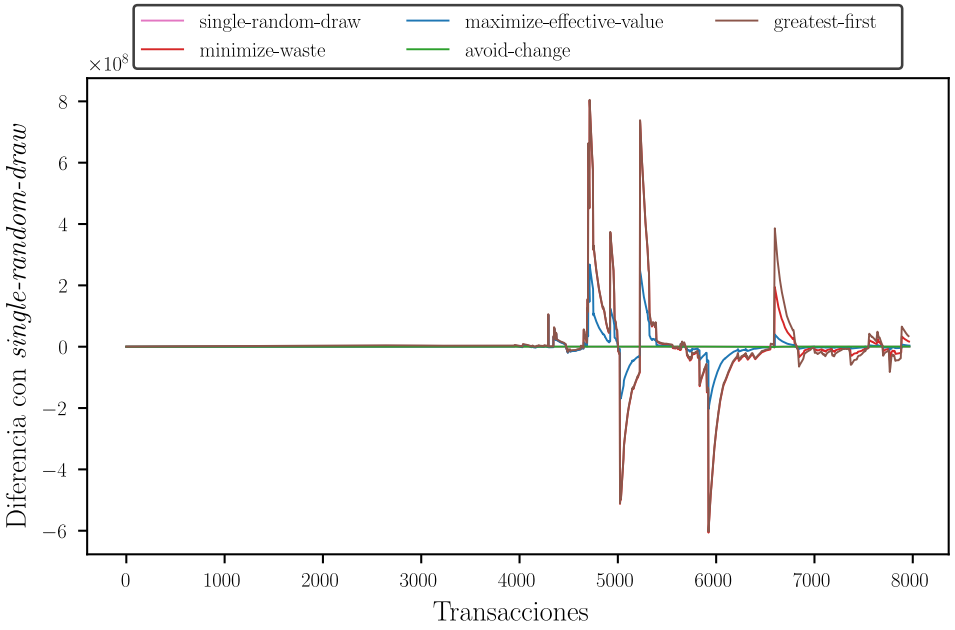


Figura 22: Balance comparado respecto de *single-random-draw*, *random-blocks*

modelo	balance final ($x10^8$ SATS)	#monedero máximo	mediana #monedero	máximo #entradas	participación
greatest first	1.5215	1340	667	449	100.00%
single random draw	1.5158	50	12	25	100.00%
maximize effective value	1.5164	119	16	77	98.63%
minimize waste	1.5183	166	31	155	99.99%
avoid change	1.5161	30	11	18	4.84%

Tabla 7: Resumen de actividad del monedero, *random-blocks*

5 Conclusión y trabajo futuro

5.1 Conclusión

En este trabajo construimos un marco teórico para abordar el problema de la selección de monedas en Bitcoin, implementamos un simulador para probar distintos modelos y realizamos un análisis comparado de los resultados. Se enfoco el problema desde la óptica de la optimización de modelos binarios lineales, utilizando la interfaz implementada en Python, *Pulp*, para interactuar con la implementación del algoritmo *Branch and Bound* del paquete *Coin-OR*.

Gracias a los resultados obtenidos del simulador pudimos demostrar que el modelo orientado a optimizar la métrica de desperdicio de una transacción es suficientemente bueno para escenarios donde la variabilidad de las tarifas es moderada, como en el caso de los escenarios utilizados. Además, pudimos comprobar que la complementación de la idea anterior junto con un ajuste estratégico del valor efectivo del cambio producido para conformar *UTxOs* de mayor utilidad futura no genera el impacto necesario en el tamaño del conjunto de *UTxOs* como para considerarlo por sobre la primer alternativa.

El código del simulador[19] implementado es abierto y esta disponible en un repositorio público con las licencias adecuadas para su uso libre y modificación.

5.2 Trabajo futuro

Debido a la extensión estipulada para el presente trabajo, quedaron fuera del mismo algunos elementos que permitirían agregar claridad al análisis de la selección de monedas. Un aspecto que el sistema implementado es capaz de procesar pero que no fue incluido en el presente trabajo, es el estudio de escenarios con múltiples pagos por transacción.

En cuanto a la manipulación voluntaria del valor efectivo de las salidas de cambio, a pesar de los resultados desalentadores, una estrategia aun no probada es la inclusión del costo de las salidas de pago futuras en la deducción del mismo. Esto permitiría que la selección con cambio busque generar salidas de cambio con valores iguales a posibles solicitudes de pago futuras. La efectividad de esta estrategia esta sujeta a la predicción de la estimación de tarifas, la cual solo puede tener sentido a corto plazo por las condiciones cambiantes del mercado de tarifas y cuando el volumen de ingresos y solicitudes de pago es lo suficientemente abundante como para generar rotación de *UTxOs* en las ventanas de tiempo estimadas.

La investigación en el área de la predicción de tarifas seria un buen complemento en general, para cualquier escenario de simulación de la selección de monedas.

Hay que tener en cuenta que el simulador creado para este trabajo fue construido como una plataforma que permite su extensión y adaptación para el desarrollo de otras investigaciones aplicadas a la selección de monedas. Por ejemplo, agregando nuevos tipos de *UTxOs*, incorporando su tiempo de vida para utilizarlo como criterio en algún modelo

[19] `https://github.com/csralvall/tx-optimizer`

o incluyendo métricas y heurísticas de privacidad[2, 20] para comparar los distintos algoritmos.

Además, gracias a que el simulador utiliza la interfaz de Pulp, es posible acceder a otros solvers sin muchos cambios en el código. Esto permite experimentar con distintos algoritmos utilizados en *ILP*, más cercanos al estado del arte, y evaluar la conveniencia de utilizar uno u otro para este problema en particular.

Finalmente, la existencia de forma pública de los monederos de algunas casas de cambio[20] deja abierta la posibilidad a la construcción de escenarios de mayor realismo y también a la evaluación del desempeño de dichos monederos mediante la comparación con los modelos utilizados en nuestro simulador.

[20] `https://www.binance.com/en/proof-of-reserves`

Bibliografía

[1] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. 2008, p. 9.

[2] LaurentMT. *Transaction Entropy*. Github Gist. URL: https://gist.github.com/LaurentMT/e758767ca4038ac40aaf.

[3] Sergi Delgado-Segura et al. "Analysis of the Bitcoin UTXO Set". In: *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers*. Nieuwpoort, Curaçao: Springer-Verlag, 2018, pp. 78–91. ISBN: 978-3-662-58819-2. DOI: 10.1007/978-3-662-58820-8_6. URL: https://doi.org/10.1007/978-3-662-58820-8_6.

[4] Thaddeus Dryja. "Utreexo: A dynamic hash-based accumulator optimized for the Bitcoin UTXO set". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 611. URL: https://eprint.iacr.org/2019/611.

[5] Svetlana Abramova and Rainer Böhme. "Your Money or Your Privacy: A Systematic Approach to Coin Selection". In: *Cryptoeconomic Systems' 20* (2020).

[6] Stephen P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[7] Iain Dunning, Stuart Mitchell, and Michael O'Sullivan. *PuLP: A Linear Programming Toolkit for Python*. https://optimization-online.org/?p=11731. Sept. 2011. URL: https://coin-or.github.io/pulp/.

[8] R. Lougee-Heimer. "The Common Optimization INterface for Operations Research: Promoting Open-Source Software in the Operations Research Community". In: *IBM J. Res. Dev.* 47.1 (Jan. 2003), pp. 57–66. ISSN: 0018-8646. DOI: 10.1147/rd.471.0057. URL: https://doi.org/10.1147/rd.471.0057.

[9] Gregory Maxwell. *User:Gmaxwell/Coin Selection*. May 2012. URL: https://en.bitcoin.it/wiki/User:Gmaxwell/coin_selection.

[10] Mark Erhardt. *An Evaluation of Coin Selection Strategies*. 2016.

[11] Mark Erhardt. *What is the Waste Metric?* May 2022. URL: https://murch.one/posts/waste-metric/.

[12] Jameson Lopp. *The challenges of optimizing unspent output selection*. Feb. 2015. URL: https://blog.lopp.net/the-challenges-of-optimizing-unspent-output-selection/.

[13] Edsko De Vries. *Self Organisation in coin selection - IOHK blog*. July 2018. URL: https://iohk.io/en/blog/posts/2018/07/03/self-organisation-in-coin-selection/#.

[14] Daniel J. Diroff. *Bitcoin Coin Selection with Leverage*. Tech. rep. 2019, p. 15.

[15] Mark Erhardt. *CoinSelectionSimulator*. Sept. 2014. URL: https://github.com/murchandamus/CoinSelectionSimulator.

[16] Daniel J. Diroff. *Bitcoin-Transaction-Optimization/transaction-optimizer*. Mar. 2019. URL: https://github.com/akvelon/Bitcoin-Transaction-Optimization/tree/master/transaction-optimizer.

[17] Andrew Chow and Mark Erhardt. *coin-selection-simulation*. Mar. 2022. URL: https://github.com/achow101/coin-selection-simulation.

[18] Miguel Campercholi, Diego J. Vaggione, and D. Martín Vilela. *Computabilidad y logica*. Cátedra de Lógica, Lenguajes Formales y Computabilidad, Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Córdoba. URL: https://drive.google.com/file/d/12ttAnnmQbHYzrZItMBDpQUCUu4mwccli/view.

[19] Nicola Atzei et al. "A Formal Model of Bitcoin Transactions". In: *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 – March 2, 2018, Revised Selected Papers*. Nieuwpoort, Curaçao: Springer-Verlag, 2018, pp. 541–560. ISBN: 978-3-662-58386-9. DOI: 10.1007/978-3-662-58387-6_29. URL: https://doi.org/10.1007/978-3-662-58387-6_29.

[20] Simin Ghesmati et al. "Unnecessary Input Heuristics and PayJoin Transactions". In: *HCI International 2021 - Posters*. Ed. by Constantine Stephanidis, Margherita Antona, and Stavroula Ntoa. Cham: Springer International Publishing, 2021, pp. 416–424. ISBN: 978-3-030-78642-7.