Extensión y optimización del algoritmo "Popcorn Void Finder" para su aplicación a relevamientos cosmológicos observacionales

Alvaro Roy Schachner[1,2] alvaro.schachner@mi.unc.edu.ar, Juan Bautista Cabral[1,3] jbcabral@unc.edu.ar, Carlos Mauricio Correa[4] carlos.correa.fayn@gmail.com, Dante Javier Paz[1,2] dpaz@unc.edu.ar, and Andrés Nicolás Ruiz[2] andres.ruiz@unc.edu.ar

[1] Facultad de Matemática, Astronomía, Física y Computación
[2] Instituto de Astronomía Teórica y Experimental
[3] Comisión Nacional de Actividades Espaciales
[4] Max Planck Institute for Extraterrestrial Physics

Resumen Los vacíos cósmicos (voids) son vastas regiones subdensas en el espacio, constituyendo las estructuras observacionales más grandes del Universo. Como tales, codifican información clave acerca de la geometría y la historia de expansión del Universo, razón por la cual son de interés para la Cosmología. En este sentido, Popcorn Void Finder (PVF) es un algoritmo de identificación de voids que se presenta en dos modos: *esférico* y *popcorn*. PVF se desarrolló inicialmente sobre simulaciones cosmológicas, permitiendo así ignorar los efectos sistemáticos presentes en los catálogos astronómicos reales.
En este trabajo proponemos abordar el proceso de adaptación del modo esférico de PVF para su uso en catálogos observacionales, tomando en consideración las buenas prácticas de la ingeniería en un contexto de necesidades de software de alta calidad, rendimiento y disponibilidad.

Keywords: HPC – LSS – reingeniería – astronformática

Extension and optimization of the "Popcorn Void Finder" algorithm for its application to real cosmological surveys

Abstract. Cosmic voids are vast underdense regions in space, constituting the largest observational structures in the Universe. As such, they encode key information about the geometry and the accelerated expansion of the Universe, which is why they are an object of interest to Cosmology. In this sense, Popcorn Void Finder (PVF) is an algorithm designed to identify such voids which presents itself in two flavours: *spheric* and *popcorn*. PVF was initially developed on cosmological simulations, thus

allowing one to ignore the systematic effects present in real astronomical catalogs.
In this proposal we adress the adaptation of PVF's spherical phase for use in observational catalogs, taking good engineering practices into consideration in a context of high quality, performance and availability of software requirements.

Keywords: HPC – LSS – reingeneering – astronformatics

1. Introducción

Los vacíos cósmicos (voids) son vastas regiones subdensas en el espacio. Al comprender la mayor parte del volumen del universo, constituyen las estructuras observacionales más grandes. Es por ello que sus propiedades físicas contienen información clave acerca de la historia de expansión y geometría del Universo. Actualmente, el estudio de los voids está recibiendo especial atención en vista de la nueva generación de relevamientos espectroscópicos de galaxias, los cuales cubrirán un volumen sin precedentes, lo que permitirá estudiar la evolución del Universo con gran precisión (Correa, 2023).

Consecuentemente, la investigación en algoritmos de detección de voids es una tarea muy útil para el avance de la cosmología observacional moderna, ya que desarrollar nuevos métodos permite extraer mayor información de los datos, comparar de manera justa las predicciones teóricas con las observaciones y asegurar la robustez de los resultados científicos. Diferentes algoritmos son apropiados para distintos tipos de datos cosmológicos y pueden tener variada eficiencia computacional y sensibilidad a efectos sistemáticos. Así, con los nuevos grandes relevamientos cosmológicos, con enormes volúmenes de datos, es útil tener una variedad de algoritmos que sean veloces, escalables y capaces de manejar los desafíos observacionales; y nos permita entender mejor la estructura a gran escala del universo, restringir parámetros cosmológicos, estudiar la energía oscura y la formación de galaxias en ambientes de baja densidad.

En este sentido, "Popcorn Void Finder" (PVF) (Paz et al., 2023) es un *toolkit* de identificación de voids en el estado del arte de código abierto [1] [2] bajo la licencia MIT, desarrollado en C++. PVF inicialmente identifica "voids esféricos" utilizando un método derivado de otra implementación (Ruiz et al., 2015). Pero luego, se agregan esferas iterativamente sobre la superficie de cada void esférico, permitiendolos expandirse y adoptar una forma más libre, como los granos de maíz que explotan y forman el "pochoclo" (popcorn).

PVF fue desarrollado y testeado sobre múltiples conjuntos de datos de prueba, en los que destacan compilaciones de datos de materia oscura y halos de

[1] https://gitlab.com/dante.paz/popcorn_void_finder

[2] Este repositorio recibe actualizaciones tardías de otro código privado donde se realizan cambios experimentales. En particular, no se encuentran las contribuciones realizadas en este trabajo; sin embargo serán incluidas en una futura publicación.

materia oscura de la simulación MillenniumXXL (Angulo et al., 2012); por lo que en su estado actual, la implementación está diseñada para trabajar con las condiciones típicas de las simulaciones cosmológicas, lo que permitió evitar la necesidad de tratar con los efectos sistemáticos presentes en los *surveys*.

2. Popcorn Void Finder

La Cosmología basada en los voids se fundamenta en la medición de sus propiedades estadísticas, las cuales se contrastan con predicciones teóricas derivadas de principios físicos (Correa, 2023).

Durante los años se han desarrollado múltiples métodos de identificación de voids en base a algún criterio con fundamento teórico, razón por la cual son reconocidos como *identificadores de voids*. Existen múltiples identificadores de voids en la literatura, cada uno con su definición propia de void basada en alguna propiedad fundamental de los voids (Neyrinck, 2008; Sutter et al., 2015; Ruiz et al., 2015). El identificador de interés para este trabajo es el conocido como *Popcorn Void Finder* (PVF) (Paz et al., 2023), sobre el cuál hablaremos luego de introducir otra clase de identificadores.

2.1. Identificadores esféricos y sus limitaciones

Los identificadores esféricos operan bajo la suposición de que los voids pueden aproximarse, en términos generales, a una forma esférica. Este enfoque simplifica el análisis de su geometría, permitiendo describirlos de manera eficiente mediante parámetros básicos como su centro y su radio. Aunque en la realidad los voids no siempre presentan formas perfectamente esféricas debido a su evolución dinámica y las interacciones gravitacionales con el entorno circundante, su simplicidad es uno de sus principales beneficios. Los modelos esféricos han sido ampliamente utilizados para explorar propiedades estadísticas de los voids, como su distribución de tamaños, su evolución temporal y su relación con el entorno cósmico (Paz et al., 2013; Ruiz et al., 2015; Correa, 2023).

La aproximación de los identificadores esféricos resulta útil para estudios iniciales y modelados teóricos; sin embargo, esto también puede introducir limitaciones al no capturar completamente la diversidad de formas y complejidades que presentan los voids en simulaciones y observaciones. Como resultado se obtienen voids esféricos que no cubren regiones sub-densas en su completitud, o en múltiples voids esféricos asociados a una única región sub-densa, fenómeno denominado *fragmentación*, y esta limitación es la que busca atacar Popcorn.

2.2. Metodología Popcorn

Motivado por la simplicidad de los identificadores esféricos, PVF es un identificador de voids que emplea un método que evita los problemas subyacentes encontrados en los identificadores esféricos. Esto se logró definiendo al void como

una *forma libre* que sea lo suficientemente flexible como para prevenir el problema de la fragmentación. Como resultado, se obtuvo una generalización de los identificadores esféricos: en lugar de utilizar una esfera para cubrir una región, el enfoque de PVF continuaría agregando esferas recursivamente para rellenar estas regiones.

El método PVF opera principalmente mediante dos parámetros clave: el rango radial de las esferas que contienen los voids ($[r, R]$) y el umbral de contraste máximo de densidad integrada (Δ_v) que determina si una región es aceptada como void. Su funcionamiento opera a modo de *pipeline* [3], donde el primer proceso realiza una identificación de voids esféricos sobre una distribución espacial de *trazadores* [4], bajo los parámetros previamente especificados. Como resultado, se obtiene un catálogo de voids esféricos que sirve como entrada al corazón del algoritmo Popcorn que, a grandes rasgos, consiste de cuatro etapas:

Colocación de semillas Sobre la superficie de cada void esférico, se distribuyen semillas de manera uniforme.

Expansión Cada semilla se expande individualmente, buscando el máximo radio posible para el que el volumen combinado del par de esferas esté por debajo del límite de contraste Δ_v. Una vez que todas las semillas han sido expandidas, se selecciona únicamente la esfera que logró el mayor volumen.

Iteración Se repite el proceso de colocación y expansión de semillas sobre la superficie del par, y las uniones subsiguientes de esferas, permitiendo la expansión progresiva del void. El proceso termina cuando ya no es posible agregar una nueva esfera cuyo radio supere el radio mínimo.

Limpieza por superposición Se aplica una limpieza que solo elimina todos los voids Popcorn superpuestos.

La implementación de algoritmos complejos no es una tarea trivial. Esta dificultad se acentúa cuando la metodología detrás del algoritmo no está completamente definida o evoluciona constantemente. El cambio constante tiene un impacto directo sobre la calidad del código y cuando este proceso no se abarca con precaución y de forma sistemática, su calidad se degrada, resultando en un código muy difícil de mantener (Tripathy et al., 2014).

Si bien, el objetivo principal de este trabajo consistía principalmente en la expansión de PVF, un análisis estructural sobre el código reveló que un proceso de reingeniería sobre el código existente sería ampliamente útil para posteriormente abarcar dicha expansión.

3. Reingeniería y adaptacion a datos de *surveys*

Con el conocimiento recientemente adquirido de PVF, vamos a proceder con el proceso de desarrollo de este trabajo, donde describiremos en detalle las modificaciones que implementamos, los desafíos que enfrentamos y los beneficios introducidos por estos cambios.

[3] https://en.wikipedia.org/wiki/Pipeline_(software)

[4] En este contexto llamamos "trazador" a un elemento que posee una ubicación en el espacio pero no asume ninguna identidad en particular.

3.1. Estructura del código

El funcionamiento de PVF está dividido en cuatro ejecutables que comparten una gran parte del código bajo una arquitectura de *pipeline*.

1. `svf`: Identificación de voids esféricos sobre un conjunto de trazadores.
2. `popcorn`: Búsqueda de voids Popcorn sobre un conjunto de voids esféricos.
3. `compute_intersecs`: Detección de voids Popcorn solapados.
4. `clean_duplicates`: Limpieza de voids Popcorn solapados.

Inicialmente, estos ejecutables se encontraban en el mismo directorio que contiene todo el código común entre ellos. Para facilitar la comprensión del software, decidimos designar un directorio exclusivo a cada ejecutable con todas las componentes relevantes en su contexto.

Adicionalmente, PVF cuenta con dos librerías auxiliares, una destinada a la interpretación sintáctica de los parámetros de la simulación, originalmente llamada "`configuration_lib`"; y la otra dedicada a la lectura y escritura de archivos de entrada y salida, con el nombre "`io_lib`" (posteriormente renombradas a "`libconfiguration`" y "`libio`" respectivamente).

En este trabajo optamos por revisar `libconfiguration`, la cual solo se encargaba de la lectura del archivo de configuración, dejando la interpretación de los datos obtenidos como tarea para el usuario. Esto en un software con cuatro ejecutables significa escribir código que interpreta configuraciones cuatro veces, empeorando mucho la mantenibilidad al momento de requerir cambios en los parámetros; por esta razón nos interesa reducir la cantidad de código que debe duplicarse. Nuestra decisión final fue la de re-escribir toda la librería, simplificando ampliamente el modo de uso, e incluyendo la interpretación junto con la lectura de datos para garantizar consistencia en una misma línea de ejecución.

El último cambio significativo fue el de asociar un grupo de módulos cuyo objetivo combinado es computar volumen de una unión de esferas solapadas en una nueva librería auxiliar "`libarvo`".

Todos estos cambios en conjunto resultaron en una mejor división lógica de las unidades de código, facilitando así la comprensión de las distintas componentes que ponen a PVF en funcionamiento.

3.2. Funcionalidad opcional

PVF fue diseñado con soporte de catálogos en múltiples formatos; entre ellos, es posible utilizar HDF5 (Folk et al., 2011). Aunque su uso no es obligatorio, la presencia de la librería HDF5 ha sido un requisito necesario para compilar PVF; esto resulta problemático, ya que obliga al usuario a instalar una biblioteca que no necesariamente utilizaría. Esto motivó el objetivo de hacer el soporte de HDF5 opcional.

La clave para lograr esto reside en el Preprocesador de C (CPP) [5], una componente fundamental de los lenguajes de familia C que extiende significativamente

[5] https://en.cppreference.com/w/c/preprocessor

sus capacidades. Esta poderosa herramienta nos provee las directivas `#ifdef` e `#ifndef`, que nos permiten incluir o excluir código en base a una condición arbitraria.

La implementación de esta idea consiste en realizar dos definiciones de cada función dependiente de HDF5. Una de ellas implementaría la funcionalidad *canónica*, que utilizaría la librería HDF5 con normalidad; mientras que la otra produciría un mensaje de error y abortaría la ejecución. Estas funciones luego se encerrarían entre estos bloques `#ifdef` eligiendo un nombre como condición de activación (como por ejemplo `USE_HDF5`).

Lo único que queda es decidir cuál de las definiciones utilizar. La mayoría de los compiladores famosos permiten definir nombres al CPP por medio de argumentos; en tales casos `-DUSE_HDF5`, activaría la condición `USE_HDF5`, produciendo la versión de HDF5; caso contrario se produce una función que simplemente genera un error.

3.3. *Bug hunting*

Buscar y solucionar *bugs* activamente no era parte de los objetivos de este trabajo; sin embargo, durante el desarrollo encontramos problemas importantes en el código original que requerían intervención.

Durante el desarrollo de este trabajo realizamos muchos *tests* para comprobar que los cambios aplicados no afectaran la correctitud general del algoritmo. En una de las primeras de estas pruebas, cometimos el error de proveer un camino incorrecto a un archivo de entrada, y aunque el programa continuó su ejecución, terminó en un ciclo infinito, evidenciando una falta de verificación de errores. Esto motivó una revisión exhaustiva del código para incorporar validaciones adecuadas, especialmente sobre entrada y salida, informar errores al usuario y abortar la ejecución si fuese necesario.

De momentos, las pruebas se realizaron sobre diferentes computadoras manteniendo los mismos parámetros de entrada, con la espera de resultados consistentes. Sin embargo, uno de los sistemas fallaba sistemáticamente. Un diagnóstico reveló que el problema se debió al uso incorrecto de una función de la librería MPI (Message Passing Interface Forum, 2023). Aunque la implementación de OpenMPI (Gabriel et al., 2004) maneja este caso de forma implícita, MPICH (MPICH, 1992) no lo hace, provocando un error.

3.4. Migración a datos de *surveys*

Hasta ahora hemos presentado qué son los voids, por qué son importantes y algunos identificadores en la literatura. Sin embargo, no discutimos explícitamente sobre el dominio de búsqueda los identificadores de voids. La intuición podría dictar que estos operan sobre una distribución de puntos en el espacio, lo cual no es incorrecto; sin embargo, en este contexto astronómico puede haber condiciones que limitan las asunciones que podemos realizar sobre estos espacios.

Para entender por qué esto podría suponer un problema, vamos a hablar brevemente sobre las dos grandes fuentes de datos en la astronomía: las simulaciones y las observaciones.

Las simulaciones numéricas del universo ofrecen acceso completo y preciso a las propiedades físicas de la materia en el contexto de una teoría física asumida (Angulo et al., 2012; Springel et al., 2021). Por el contrario, los catálogos observacionales son un set de datos independientes de un modelo o teoría física, que provienen de grandes relevamientos del cielo (*surveys*) y están sujetos a efectos sistemáticos como incompletitud, errores de medición y sesgos de selección, derivados de las limitaciones instrumentales (York et al., 2000). Estas características imponen restricciones que requieren un tratamiento específico que no fué tomado durante el desarrollo de PVF. Como última etapa de este proceso abordaremos la adaptación del algoritmo esférico de PVF para su soporte sobre *surveys*.

En los catálogos simulados que manipulamos, el cálculo del volumen de una región esférica de radio r se calcula analíticamente (Ecuación 1).

$$V = \frac{3}{4} \cdot \pi \cdot r^3 \tag{1}$$

Sin embargo, cuando se trabaja con catálogos observacionales, la incompletitud de los datos y los efectos de selección pueden sesgar significativamente el cálculo del volumen, ya que la región esférica puede no estar completamente contenida dentro de la sección observada. Por estas razones, es necesario hacer uso de métodos numéricos.

Este es un tipo de problemas donde el método Monte Carlo [6] destaca especialmente. La solución consiste en utilizar un catálogo secundario con trazadores generados con números pseudo-aleatorios (*randoms*), los cuales estarían concentrados de manera uniforme solo en las regiones contempladas por el catálogo real de galaxias. De esta forma, con la densidad global del catálogo auxiliar Δ_r, es posible calcular el volumen integrado V_i de una región que contiene n_r *randoms* (Ecuación 2).

$$V_i = \frac{n_r}{\Delta_r} \tag{2}$$

A modo de ejemplo, la Figura 1 ilustra cómo este procedimiento nos permite aplicar la Ecuación 2 en base al conteo de elementos que se encuentran dentro de la región aceptada.

Con el fin de mantener compatibilidad entre ambos tipos de catálogos, incorporamos un nuevo parámetro que permite especificar si el catálogo de entrada corresponde a una observación o a una simulación. En el caso de trabajar con observaciones, también requerimos proporcionar un catálogo adicional que contenga los *randoms* correspondientes.

Lo último que nos queda es implementar el método Monte Carlo. Para este fin, comenzamos por duplicar el algoritmo original que calcula volúmenes de manera exacta. Luego, identificamos todas las instancias de cálculo de volumen

[6] https://en.wikipedia.org/wiki/Monte_Carlo_method

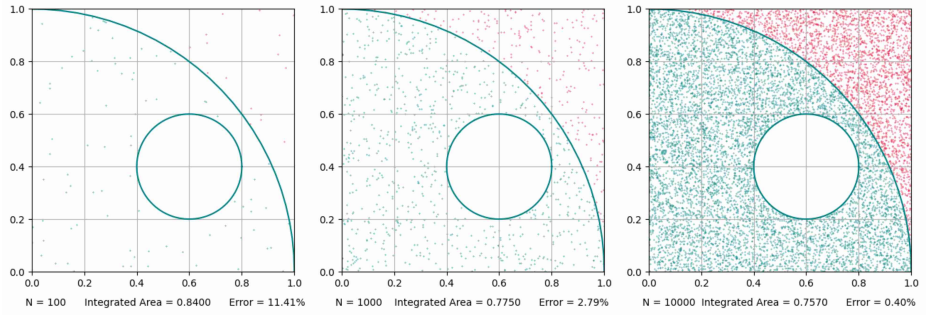

Figura 1. Estimación de la superficie de una porción circular agujereada de área $A \approx 0{,}7539$ con distintos tamaños de muestreo N, resultando en $\Delta_r = N$.

y las reemplazamos por un conteo de los *randoms*, aprovechando los algoritmos *Nearest Neighbour Search* que ya estaban en el código y permiten reducir la necesidad de contar *randoms* individualmente en algunas instancias.

Con este conteo terminado, solo resta reemplazar el uso de la Ecuación 1 por la Ecuación 2 permitiendo estimar el volumen de cualquier región independientemente de cualquier fenómeno sistemático en el que se encuentra.

4. Resultados

Las modificaciones propuestas en este trabajo se pueden dividir en dos fases: aplicación de ingeniería y adaptación a *surveys*, resultando en tres estados distintos del código para comparar, a los que llamaremos *original* para el código en el estado inicial, `dev` para el que se le aplicaron los cambios de reingeniería, y `survey` para el adaptado con soporte de *surveys*.

4.1. Análisis de ingeniería

Para este análisis, hicimos uso de la herramienta `lizard` [7], que nos permite calcular el Número de Complejidad Ciclomática (CCN) (McCabe, 1976) junto al Número de Líneas de Código (NLOC) total y promedio por función, produciendo los resultados que se pueden ver en la Tabla 1.

Es remarcable que la adición del método Monte Carlo en `survey` resultó en un aumento del CCN lo suficientemente significativo como para superar al CCN original. Sin embargo, un cambio leve en el promedio no nos da mucha información sobre los módulos afectados, esto requiere un análisis más profundo. Haciendo una revisión detallada de las *warnings* generadas por `lizard`, comprobamos que una de las mayores fuentes del CCN elevado está en las versiones modificadas de las funciones que componen el método de identificación esférica.

[7] https://github.com/terryyin/lizard

Versión	NLOC Total	NLOC Promedio	CCN Promedio
original	7015	29,0	5,1
`dev`	6570	28,0	4,9
`survey`	7157	29,7	5,2

Tabla 1. Análisis de CCN y NLOC de PVF.

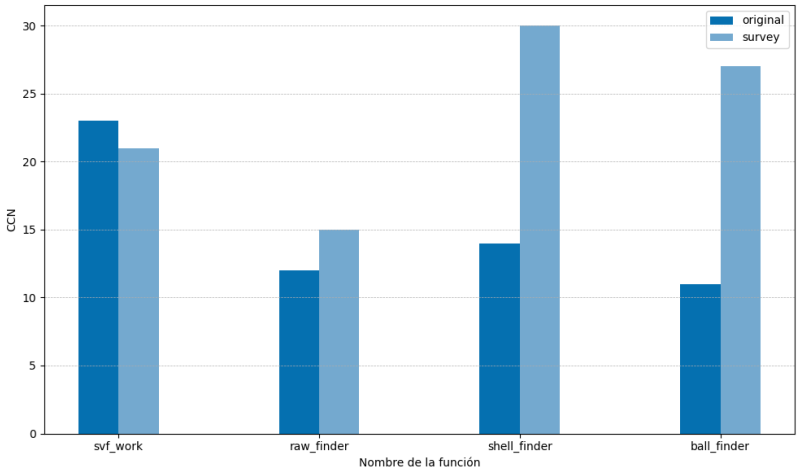

Figura 2. Comparación del CCN entre el método de identificación esférica para simulaciones y el método de identificación para *surveys*.

Como se puede ver en la Figura 2, el CCN aumentó dramáticamente en las funciones `shell_finder` y `ball_finder` en la versión `survey`. En estos casos, implementar el conteo de *randoms* requirió duplicar gran parte del cuerpo de las funciones para hacer uso de los algoritmos de búsqueda de vecinos. Por otro lado, en la función `svf_work` si se logró abstraer este conteo, tanto para los trazadores como para los *randoms*, resultando en menos líneas de código que presentaban muchos ciclos y condicionales, esto en efecto redujo el CCN.

4.2. Voids esféricos con Monte Carlo

Para validar el método de identificación sobre *surveys*, vamos a realizar pruebas sobre un *mock* con la forma de un agujero Gaussiano en una caja de volumen $V = 20^3$, con número de trazadores $N_{trac} \approx 10^6$. Aunque estas pruebas no sean sobre un catálogo real, es posible imitar las condiciones ideales de las simulaciones para que el algoritmo aproximado se comporte como queremos: al no haber ningún espacio incompleto, o cubierto por funciones de selección, solo se necesita un catálogo de *randoms* distribuidos uniformemente en el espacio. Realizar este tipo de experimentos sobre un *mock* nos permitirá evaluar la capacidad del

algoritmo para *surveys* bajo condiciones controladas, lo cual no es posible con datos observacionales.

El algoritmo para *surveys*, que calcula el volumen de manera estimada utilizando Monte Carlo, detectó tres voids esféricos en todas las pruebas (Figura 3).

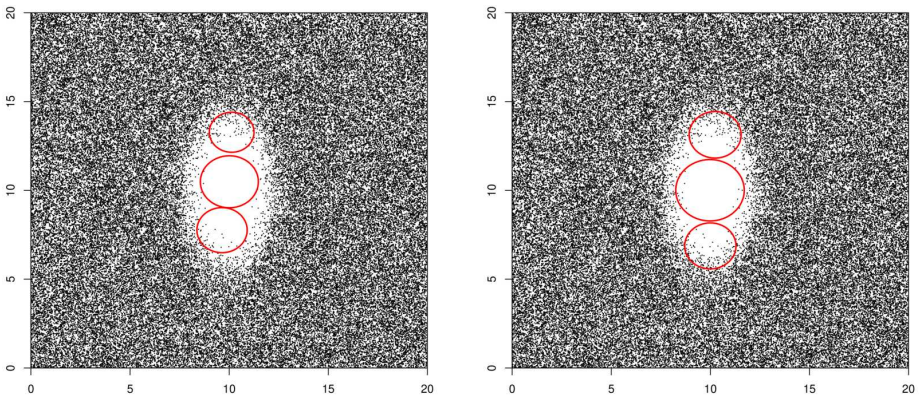

Figura 3. Identificación de voids esféricos con estimación de volumen Montecarlo utilizando $2 \cdot N_{trac}$ *randoms* (izquierda) y $10 \cdot N_{trac}$ *randoms* (derecha).

Si bien estos resultados se ven convincentes, es remarcable que se encontraron menos voids que con el algoritmo original, estos siendo cuatro esferas en los confines laterales de la región subdensa. La ausencia de los voids puede simplemente haber sido por una cantidad insuficiente de *randoms*, pero a este punto decidimos optar por otro tipo de experimento para investigar por qué sucedía esto. Una componente muy importante de los algoritmos esféricos en PVF es la "limpieza por superposición", que elimina todos los voids superpuestos. A modo de prueba, eliminamos esta fase, efectivamente manteniendo todos los voids esféricos que el algoritmo con aproximación Monte Carlo logró encontrar. Los resultados se pueden ver en la Figura 4.

Notemos que ahora toda la región "vacía" se encuentra pintada de rojo. Esto nos dio suficiente evidencia para convencernos de que la implementación del algoritmo con volumen aproximado es, en primera instancia, correcta.

4.3. Desempeño

La adición de un catálogo de *randoms* al método para *surveys* agrega muchas variables libres que podrían afectar el rendimiento de manera significativa, lo que dificulta hacer una comparación directa entre ambos métodos. En su lugar, hicimos una comparación de los tiempos de ejecución de los resultados obtenidos en la Subsección 4.2, para ver si encontrábamos alguna relación entre los tiempos

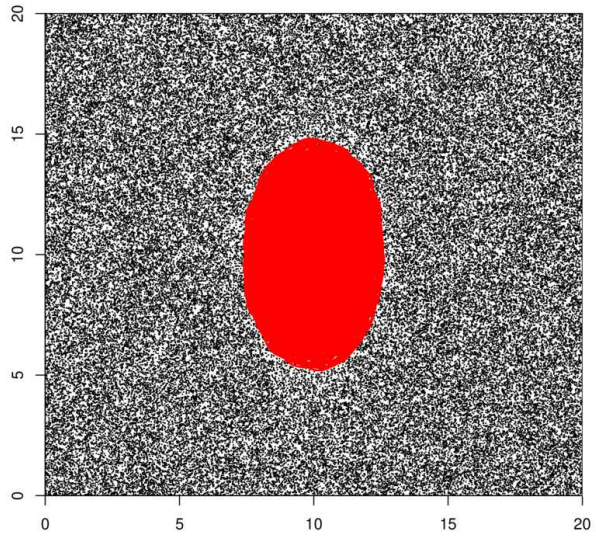

Figura 4. Identificación de voids esféricos con estimación de volumen Montecarlo utilizando $2 \cdot N_{trac}$ randoms, sin limpieza de esferas solapadas.

de ejecución. Las pruebas se realizaron en la computadora `zx81` (FAMAF–UNC) con las especificaciones de la Tabla 2.

Sistema Operativo	Debian trixie/sid Linux 6.12.9-1 (2025-01-10)
CPUs	2 x Intel(R) Xeon(R) E5-2680 2.40GHz 28C / 56 N
Memoria	8 x 16 GiB (256 GiB Total)
Compilador	`gcc@14.2.0`
MPI	`mpich@4.2.1`

Tabla 2. Especificaciones y entorno de zx81.

Recordemos que los catálogos auxiliares contaron con número de *randoms* múltiplos de $N_{trac} \approx 10^6$ distribuidos de manera uniforme en el espacio. Los resultados de ejecución se pueden ver en la Figura 5.

Es destacable que, entre la ejecución del algoritmo original y la primera ejecución del algoritmo para *surveys*, el tiempo de ejecución aumenta casi diez veces. Sin embargo, los tiempos de las ejecuciones sucesivas del algoritmo para *surveys* aumentaron de forma controlada, aparentemente tendiendo a una recta.

Para evaluar el uso de paralelismo también realizamos pruebas de escalado fuerte (Amdahl, 1967). Para esto, optamos por fijar el tamaño de problema a $2 \cdot Ntrac$ porque supone una cantidad de tiempo suficiente para despreciar el tiempo de *IO* e inicialización de recursos para todos los casos. Con respecto a la cantidad de núcleos, nos limitamos por elegir valores acotados por la cantidad

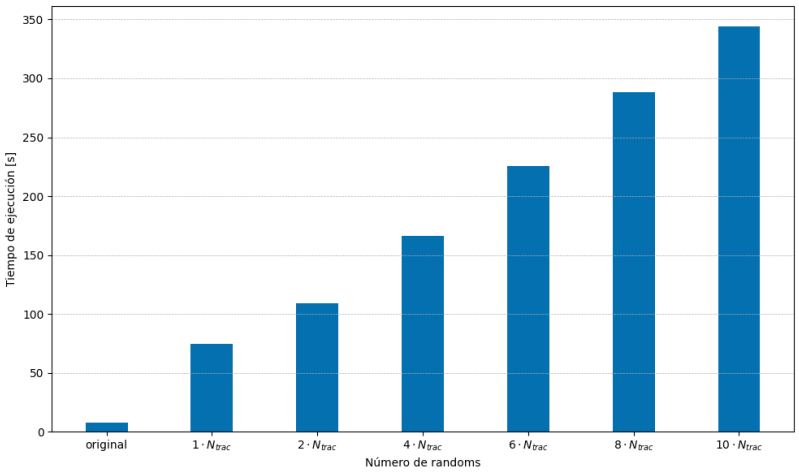

Figura 5. Comparación directa de tiempos de ejecución entre el método para simulaciones y el método para *surveys* en múltiples configuraciones.

física de núcleos que ofrece `zx81`, que en este caso serían todas las potencias de dos menores a 32, junto con 28, que es el número total de núcleos. De esta forma podemos calcular el *speedup* $S(N)$ de escalado fuerte del código en función de esta cantidad de núcleos. Adicionalmente, realizamos un ajuste al *speedup* (Figura 6) como lo sugerido en Xavier et al. (1998) que nos muestra la *eficiencia* del escalado fuerte.

Recordemos que la ley de Amdahl, escrita en términos de las secciones serial–paralela del código, nos ofrece una cota superior $A(N)$ al *speedup*. Aplicándola sobre los resultados obtenemos la relación $S(28) \leq A(28)$, lo cual nos permite estimar que en este caso particular el 99,4 % del código ejecuta en paralelo. Estos resultados en conjunto nos dan un buen indicio de que el paralelismo es eficiente.

5. Conclusiones y trabajo a futuro

La evolución desde los identificadores esféricos hasta PVF permitió representar con mayor precisión la morfología de los voids, mejorando tanto el análisis estructural como el cálculo de abundancias. Por un lado, realizamos modificaciones sobre el código original para mejorar su funcionamiento y mantenibilidad, donde reorganizamos su estructura e introdujimos otros cambios para mejorar su confiabilidad. Por otro lado, integramos el método Monte Carlo al modo esférico de PVF, permitiendo avanzar un paso en la identificación de voids Popcorn sobre catálogos observacionales.

Estos cambios fueron acompañados con un análisis cuantitativo que nos permitió evaluar el impacto del código junto con la validez de los nuevos algoritmos. Nuestros resultados indican que el método esférico adaptado para surveys es una

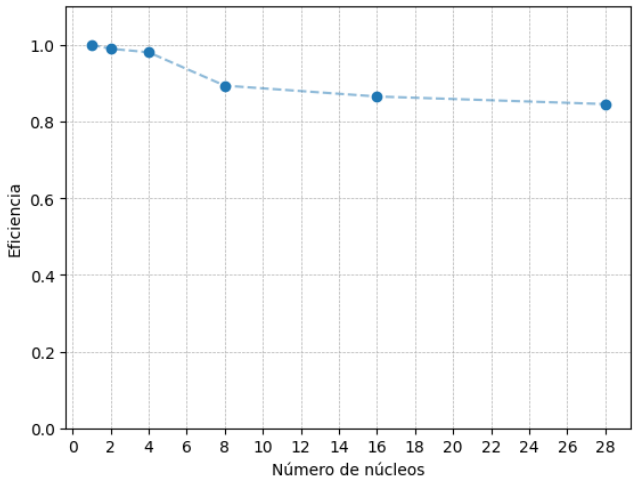

Figura 6. Eficiencia de escalado fuerte del método de identificación esférica con Monte Carlo fijando la cantidad de *randoms* a $2 \cdot Ntrac$.

alternativa viable frente a otros identificadores, y proyectamos que lo mismo pronto podrá decirse sobre la metodología Popcorn.

Si bien los identificadores de voids cumplen un rol clave, su verdadero valor surge al integrarlos con otras herramientas que permiten estudiar la evolución de los voids, su conexión con la materia y energía oscura, y su uso como trazadores cosmológicos. Esto constituye solo una etapa dentro de un pipeline más amplio, donde la identificación de voids es un paso intermedio en el análisis de la estructura a gran escala del universo. Como trabajo futuro, proponemos aplicar Monte Carlo al modo Popcorn para construir un identificador completo que funcione en surveys. Además, proponemos lanzar un *release* para lo cual no solo será necesario realizar la propuesta anterior, sino que mejorar aún mas el código para que sea más fácil de *testear*.

6. Agradecimientos

Dedico esta sección para agradecer a CONICET por otorgarme la beca doctoral. Extiendo mis agradecimientos a todos los colaboradores por su activa disposición en asistirme ante cualquier problema. Finalmente, agradezco a mi director de doctorado, Dr. Juan A. Rico por aceptarme como alumno doctoral.

Referencias

Amdahl, Gene M. (1967). “Validity of the single processor approach to achieving large scale computing capabilities”. En: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS ’67 (Spring). Atlantic

City, New Jersey: Association for Computing Machinery, págs. 483-485. ISBN: 9781450378956.

Angulo, R. E. et al. (nov. de 2012). "Scaling relations for galaxy clusters in the Millennium-XXL simulation". En: *Monthly Notices of the Royal Astronomical Society* 426.3, págs. 2046-2062.

Correa, C. M. (ago. de 2023). "Cosmic voids as cosmological laboratories". En: *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina* 64, págs. 159-165.

Folk, Mike et al. (2011). "An overview of the HDF5 technology suite and its applications". En: *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. AD '11. Uppsala, Sweden: Association for Computing Machinery, págs. 36-47. ISBN: 9781450306140.

Gabriel, Edgar et al. (sep. de 2004). "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". En: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, págs. 97-104.

McCabe, T.J. (dic. de 1976). "A Complexity Measure". En: *IEEE Transactions on Software Engineering* SE-2.4, págs. 308-320. ISSN: 1939-3520.

Message Passing Interface Forum (nov. de 2023). *MPI: A Message-Passing Interface Standard Version 4.1*.

MPICH (1992). *MPICH*.

Neyrinck, Mark C. (jun. de 2008). "ZOBOV: a parameter-free void-finding algorithm". En: *Monthly Notices of the Royal Astronomical Society* 386.4, págs. 2101-2109.

Paz, Dante J. et al. (dic. de 2013). "Clues on void evolution-II. Measuring density and velocity profiles on SDSS galaxy redshift space distortions". En: *Monthly Notices of the Royal Astronomical Society* 436.4, págs. 3480-3491.

Paz, Dante J. et al. (jun. de 2023). "Guess the cheese flavour by the size of its holes: a cosmological test using the abundance of popcorn voids". En: *Monthly Notices of the Royal Astronomical Society* 522.2, págs. 2553-2569.

Ruiz, Andrés N. et al. (abr. de 2015). "Clues on void evolution - III. Structure and dynamics in void shells". En: *Monthly Notices of the Royal Astronomical Society* 448.2, págs. 1471-1482.

Springel, Volker et al. (sep. de 2021). "Simulating cosmic structure formation with the GADGET-4 code". En: *Monthly Notices of the Royal Astronomical Society* 506.2, págs. 2871-2949.

Sutter, P. M. et al. (mar. de 2015). "VIDE: The Void IDentification and Examination toolkit". En: *Astronomy and Computing* 9, págs. 1-9.

Tripathy, P. y K. Naik (2014). *Software Evolution and Maintenance: A Practitioner's Approach*. Wiley. ISBN: 9780470603413.

Xavier, C. y S.S. Iyengar (1998). *Introduction to Parallel Algorithms*. Wiley Series on Parallel and Distributed Computing. Wiley, pág. 54. ISBN: 9780471251828.

York, Donald G. et al. (sep. de 2000). "The Sloan Digital Sky Survey: Technical Summary". En: *Astronomical Journal* 120.3, págs. 1579-1587.