The Apple File System has some support for data deduplication.
You can create a copy-on-write clone of a file, using no extra disk space
(until you start modifying either the original or the clone).

I wanted to use this for my games volume. Multiple games using the same
game engine often have many support files in common. Additionally,
multiple versions of the same game often have many game-specific files
in common.

Keeping track of this is not difficult. You simply need to maintain
a database with a file table having size and content hash columns.

But what about partial matches? For example, if you append files to a ZIP
archive, the old and new versions of the archive will have a common prefix.
When installing the new version, you could make a clone of the old one,
then copy only the non-matching parts from the source.

Keeping track of **this** gets more elaborate. You need to store hashes
of each allocation block of each file. Here's the final design:

```
    create table file (
        fileid integer
            primary key,
        size integer
            not null,
        moddate integer
            not null
    );

    create table block (
        fileid integer
            not null
            references file (fileid)
                on update restrict
                on delete cascade,
        blockix integer
            not null,
        hash integer
            not null,

        primary key (fileid, blockix)
    ) without rowid;

    create index block_hash_ix
        on block (hash, blockix);

    create temporary table newblock (
        blockix integer primary key,
```

```
        hash integer not null
    );
```

There is no file path, because macOS lets you look up files by id.
The hash values are cryptographic hashes truncated to 64 bits and
reinterpreted as integers. I picked the BLAKE3 hash function because
its standard implementation has a convenient interface for truncated
and extended results.

The newblock table is used when adding a new file. First, fill it up with
hashes computed from the source. Then, find what existing file has the
most blocks in common with the new file **and** the index of each such
block:

```
    with similar (blockix, fileid) as
        (select blockix, fileid
            from newblock
                natural join block)
    select fileid, blockix from similar
        where fileid=
            (select fileid from similar
                group by fileid
                order by count(*) desc
                limit 1)
        order by blockix;
```

This query demonstrates the importance of database analysis.
Without statistics, things get painfully slow as the database grows:

```
    QUERY PLAN
    |--SEARCH block USING PRIMARY KEY (fileid=?)
    |--SCALAR SUBQUERY 2
    |  |--SCAN block
    |  |--SEARCH newblock USING INTEGER PRIMARY KEY (rowid=?)
    |  `--USE TEMP B-TREE FOR ORDER BY
    `--SEARCH newblock USING INTEGER PRIMARY KEY (rowid=?)
```

With statistics, things remain bearable:

```
    QUERY PLAN
    |--SEARCH block USING PRIMARY KEY (fileid=?)
    |--SCALAR SUBQUERY 2
    |  |--SCAN newblock
    |  |--SEARCH block USING COVERING INDEX block_hash_ix (hash=? AND blockix=?)
    |  |--USE TEMP B-TREE FOR GROUP BY
    |  `--USE TEMP B-TREE FOR ORDER BY
    `--SEARCH newblock USING INTEGER PRIMARY KEY (rowid=?)
```

So, does all this complexity actually save any disk space? First, the
database tracking my games volume uses 729587 allocation blocks itself.
Here's a summary extracted from it:

|match|file count|total blocks|saved blocks|
---
full	259924:	9485545:	9485545:
partial	35340:	10551328:	1615241:
none	214368:	53298995:	0:

The simpler alternative approach would use a schema like this:

```
    create table file (
        fileid integer
            primary key,
        size integer
            not null,
        moddate integer
            not null,
        hash integer
            not null
    );

    create index file_hash
        on file (hash);
```

That database would use 5314 allocation blocks.

In summary, the complex approach saves 1615241 extra blocks,
using 729587-5314=724273 more blocks for its database.

1615241>724273. Win!