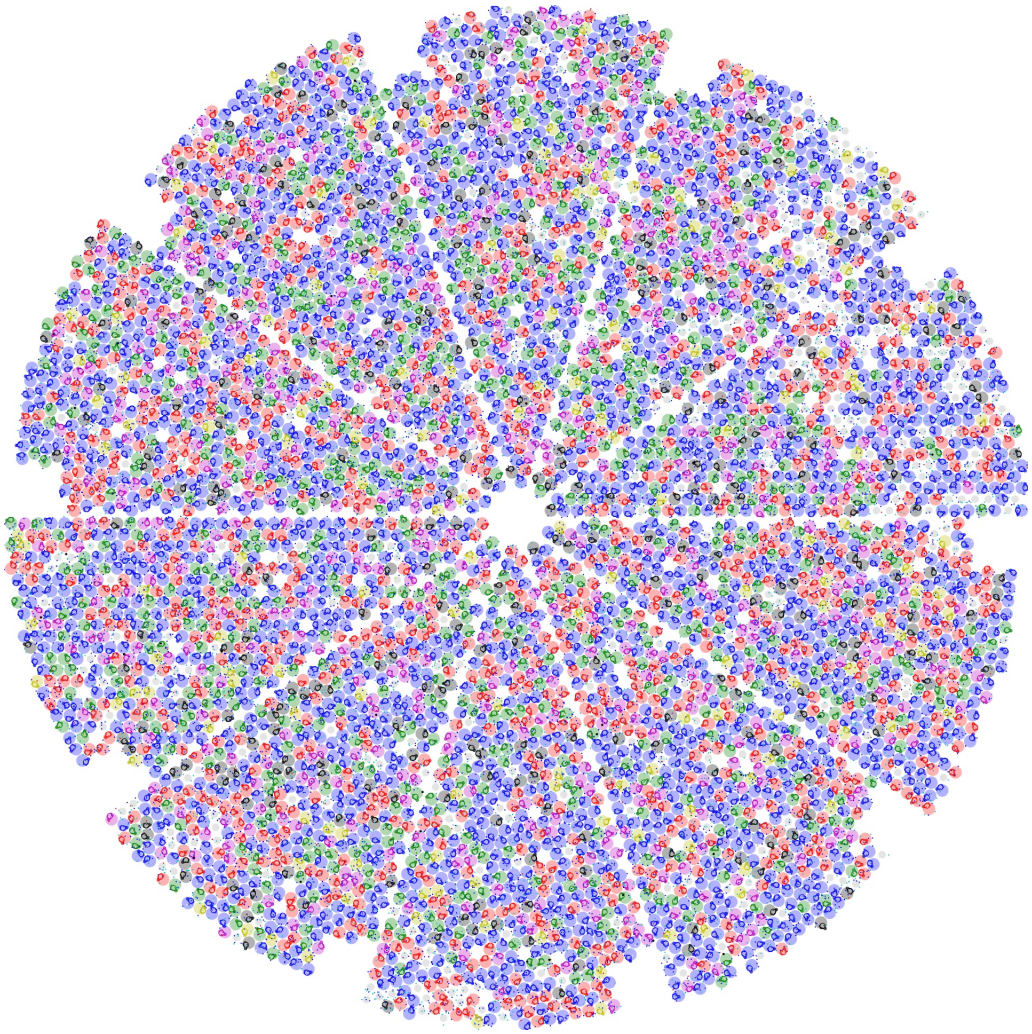
Description of fiber assignment code for Mocks in DESI experiment

Robert N. Cahn*[1] and Louis Garrigue†[2]

[1]Department of Cosmological Physics, LBNL, Berkeley
[2]Departement de physique, Ecole normale superieure, Paris

3 juin 2015



*rncahn@lbl.gov
†louis.garrigue@ens.fr

1 Introduction

Martin White developed C++ code for fiber assignment over the full 14k deg^2footprint. We first modified Martin's code to incorporate features from Bob Cahn's python code, which ran on a restricted 480 deg^2... We included the improvement and redistribution steps, which switch fiber assignments to increase the number of galaxies observed. We then adapted the code to compute assignments not globally "knowing all information on galaxies yet to be observed" but plate after plate, as in the real experiment. The samples are taken from Martin's mocks in stored on NERSC at /project/projectdirs/desi/mocks/preliminary/. From these files we create a single file containing the appropriate mix of ELG, LRG, QSO, SS (Standard Stars) and SF (Sky Fibers) using the python script in git fiberassign/bin/make_catalog_starsandsky.py. In the same place there is python code to produce a mixture of galaxies without any correlations, but with the correct $\frac{dn}{dz}$

2 In a nutshell

2.1 Input files

Found in the NERSC repository (except for Parameters file features.txt). The produced executable is `assign`. The calling sequence will look like : `./assign features.txt`. An example of how runing it on NERSC is provided in the `run` script, then call it with qsub run.

— Parameters : all parameters used in the fiber assignment are written in this file. It contains the adress of other input files
— Target DB : information, before the study, on all possible targets : (NERSC rep) /projects/projectdirs/desi/mocks/preliminary/objects_ss_sf0.rdzipn, created by `make_catalog`
— Obs DB : batabase constructed from the ongoing DESI observations after the data has been processed by the Spectroscopic pipeline. This DB has not been designed yet
— Survey tiles : file containing the positions of all the tiles to be observed in /project/projectdirs/desi/software/edison/desimodel/0.3.1/data/footprint/desi-tiles.par
— Fiber positions : locations of the positioners in the focal plane in /project/projectdirs/desi/software/edison/desimodel/0.3.1/data/focalplane/fiberpos.txt

To change the location of other input files, one can simply change it in the features file.

2.2 Output files

They are produced in a directory defined in features.txt (/project/projectdirs/desi/users/rncahn/fa_output/ for now), in the format `tile54.fits` for example for the 54th tile. There are therefore 10666 such binary files. They consist in 5000 lines (fibers) with the following columns :

— fiber : [0-4999]
— positioner : [0-4999] (not provided yet)
— number of available objects
— ID of available objects
— objtype : ELG, LRG, QSO, SKY, STDSTAR, GAL, OTHER (actually QSOLy-a, or QSOFake etc for now) (-1 if the fiber isn't assigned)
— targetid : unique target identifier to get back to target selection info
— desi target0 : 64 bit mask of targeting info (not yet)
— ra : degrees [0-360]
— dec : degrees [-90 - +90]
— xfocal : mm from center in positioner coordinate system
— yfocal : mm from center in positioner coordinate system

3 Source files

Source files are file.h and file.cpp and are in this increasing dependency order :

— misc : a home-made library of structures (and functions on them) needed to manipulate concerning datas, but independent of them. There are pair (of int), List (of int), Table, Cube, and timing, printing, string conversion, error report items.
— collision : used to compute collision checkings and build polygons

— features : stocks all useful parameters
— structs : structures of the manipulated datas and their members
— global : main high-level functions and algorithms used in the program to collect information, assign fibers and print statistics. Important ones are described further
— main : neat and quickly understandable code that sums up all steps

4 Parameters

Here are some features on input galaxies simulated catalog :

Kind	Id	Priority	Nobs	Density ($obj \cdot deg^{-2}$)
QSO Ly-α	0	1	5	50
QSO Tracer	1	1	1	120
LRG	2	3	2	300
ELG	3	5	1	2400
Fake QSO	4	1	1	90
Fake LRG	5	3	1	50
Standard Star	6	2	1	140
Sky Fiber	7	4	1	1400

TABLE 1 – Characteristics of galaxy samples as set in make_catalog_rnc.py

A number of parameters are defined in features.txt.
— parameters summed-up in the Table 1
— Npass = 5 number of passes
— MaxSS = 10 ; MaxSF = 40 maximum number of fibers assigned to SS and SF on a petal
— PlateRadius = 1.65° radius of the plate
— TotalArea = 15789.0 deg^2total area of the sky considered
— invFibArea = 700 inverse of area in deg^2accessible to a fiber (fiber density for a deg^2)
— PatrolRad = 6.0 mm maximum distance, on plate coordinates, that allows a fiber to a galaxy
— InterPlate = 200 minimal number of plates between two observations of the same galaxy
— Randomize = false randomize order of plates in making plans
— Pacman = false selects only spectrometers 0, 1, 2, 7, 8, 9 of the pacman
— Collision = true when we want to allow collisions, to compute the collision rate for instance
— Exact = true whether we want exact collision checking (exact geometry of components) or just circles
— AvCollide = 3.2 mm in case of no exact collision checking, limit distance between two galaxies for their positioners to collide (so that we have the same collision rate than with exact geometry)
— Collide = 1.98 mm minimum distance allowed on plate projection of two assigned galaxies on the same plate (optimizes collision checking)
— NoCollide = 7 mm maximum distance between two galaxies for the collision of their corresponding galaxies (optimizes collision checking)
— NeighborRad = 14.0 mm maximum distance to consider that two fibers are neighbors
— PlotObsTime, etc whether we want to plot some information into output files
— Verif = false, whether we verificate that the assignment is sane (no collision, sane mapping, etc...)

5 Classes and structures

Classes and structures are built to be independent of each other, flexible, quickly understandable, logical, and with no redundant information as much as possible.

Structure name	Meaning	Description
PP	Plate Parameters	Carries locations of fiber positions on the plate, spectrometer correspondence, and neighboring fibers information.
onplate	Plate coordinates	Used for coordinates in the focal plane in mm. The member `id` is used to give the identity of a galaxy. Onplates is vector of onplate.
plate	A plate	Locations in the sky of the tile in terms of a unit vector derived from RA and DEC. Carries also the Id of the tile, its pass, and the available galaxies it is able to reach. Plates is vector of plate, and has all information on tiles.
galaxy	A galaxy	Information on a galaxy : an `id` that corresponds to Table 1, a position in the sky (in two differents ways), and the available tile-fibers that can observe it. Gals is vector of galaxy and carries all information on galaxies.
Assignment	An entire assignment	Carries mapping of tile-fibers to galaxies, its inverse, galaxies to tile-fibers, and the cube (3d-matrix) of assigned fibers for a kind, a petal and a plate (useful for fast computations).

TABLE 2 – Classes and structures

6 Functions

In algorithms, j stands for a plate, k for a fiber, p for a petal, g for a galaxy.

6.1 Improvement functions ideas

A first assignment is done by the function simple_assign, and is the same than in BOSS experiment. Here we present the three ideas of improvement of this first assignment, which increase the number of used fibers and the quality of the assignment. It represents the body, the main purpose of our work. They need to be somewhere "orthogonal", so that they don't loose efficiency when we call one of them after an other. Those three ideas are coded into the three following functions :

FIGURE 1 – Improve

The idea is to take an unassigned fiber and to look at the galaxies it can reach and that are already assigned. We then look at the second fiber, which is observing this galaxy and try to reassign it to an other galaxy, and assign the first fiber to the first object. It is powerful because we use it when making a plan, and so the two fibers can (and are almost always) from different plates, that can be arbitrarily separated.

FIGURE 2 – Redistribute

In redistribute, we look at an assigned fiber, and try to make the observed object observed by and other fiber, which can come from a different plate. It "anneals" the assignment. In this way, it doesn't improves anything, but we then apply the improve function again, which is more efficient. The idea is the improve, then redistribute, then improve again, redistribute etc. If there is no redistribution between a new improve, the improve is almost useless.

FIGURE 3 – Improve from kind

The third idea is to take an assigned fiber that is able to reach a SF (or SS), to assign it, and then we reassign a fiber assigned to a SF. As the number of SF per petal has to remain the same as previously (40) we search the second fiber amongst the same petal of the first fiber.

6.2 Some functions in structs.cpp

`plate_dist` turns radians into mm on the focal plane, i.e. it is the plate scale as a function of angle.

`change_coords` combines a galaxy and a particular plate to give the coordinates the galaxy will have in the focal plane when observed as with this tile. It's a rotation in angular coordinates. This ought to be rigorously checked.

`find_collision` returns the fiber number of a fiber that conflicts with tiblefiber (j,k), collision checking can be with exact geometry (if Collision parameter is true) or reduced one

6.3 Collecting

`collect_galaxies_for_all` is multithreaded, and for each fiber of each tile, collects reachable galaxies. It uses kdTree and htmTree libraries written by Martin White, because it's absolutely necessary to do computations in a reasonable time with supercomputers.

`collect_available_tilefibers` computes, using the previous work, available tile-fibers for each galaxy (inverse map)

6.4 Useful sub-functions for global functions

`ok_assign_g_to_jk_nobs and ok_assign_tot` checks to see if we can assign g to the tile-fiber (j,k), according to assigning rules described further

`find_best (j,k)` finds the best reachable galaxy for this fiber, according to assignment rules

Algorithm 1 Find best(j,k)

```
Initialize a fictional galaxy with ID = −1, called best
for each galaxy available galaxy (for this fiber) g do
    if g is better (according to priority and number of observations) then
        best ← g
```

`assign_fiber(j,k)` tries to assign this fiber using find_best

`improve_fiber (j0,n,j,k)` if this fiber is unused, tries first to simply assign it, and if it doesn't work, tries to reassigning some used one (jp,kp) where $j0 \leq jp \leq j0+n$. Before : (jp,kp) - g ; (j,k) & gp free. After : (j,k) - g & (jp,kp) - gp. The power of this function lies in the fact that j and jp can correspond to different passes.

Algorithm 2 Improve_fiber(j0,n,j,k)

```
if k is not assigned then
    try to assign running assign_fiber(j,k)
    if k couldn't be assigned this way then
        initialize a set of variables jp, kp, g, gp
        for each galaxy g available to (j,k) do
            if it's possible to assign g with k then
                for each chosen tile-fibers (jp,kp) which chose g (where j0 ≤ jp ≤ j0 + n) do
                    gp ← find_best(jp, kp)
                    memorize jp, kp, g, gp if it's a better set (gp is more worthy) than previous one
        if gp ≠ −1 then
            Unassign (jp, kp) ⟷ g
            Assign (j, k) ⟷ g
            Assign (jp, kp) ⟷ gp
```

6.5 Assigning making a plan

In the simulated catalog, we know all information on objects (if a QSO is a real one for example). During the real study, one won't have access to this information prior to the observation. Thus, in the code, we have to simulate that we have this infotmation only when we have at least once observed the object.

Thus, algorithms do a plan with only information available by the catalog and by previously seen galaxies. A plan consists of an assignment on a set of consecutive tiles (from 0 to 2000 for instance). One can use algorithms on size-one plans if one wants to do an assignment plate after plate for example (which shouldn't be called a plan anymore).

The argument "next" (integer) in following functions means that we treat all next "next" plates (in the right order of tiles) in the plan we make. If it is -1, it will deal with all left plates. For example, if next plate in the program is 10 and one launch a function with $next = 100$, this function is going to do his job on plates from 10 to 110.

`simple_assign` makes a first simple assignment plan : for each fiber assign to the best available galaxy

Algorithm 3 Simple_assign(j0,n)

```
1: for each plate j from j0 to j0+n do
2:     for each fiber k in a random order do
3:         try to assign_fiber(j,k)
```

`new_assign_fibers` makes a first assignment plan trying to assign first QSO, then LRG then ELG, then trying to assign SF and SS replacing other kinds if there are not enough available fibers. This way, a QSO can't be lost because of a collision. With the current value of Collision, it's not very useful, but could be for bigger values.

Algorithm 4 New_assign_fibers(j0,n)

```
1: for each plate j from j0 to j0+n do
2:     for each petal p of this plate, in a random order do
3:         for each fiber k of this petal, in a random order do
4:             Try assign_fiber(j,k) only allowing QSO
5:         for each fiber k of this petal, in a random order do
6:             Try assign_fiber(j,k) only allowing LRG
7:         for each fiber k of this petal, in a random order do
8:             Try assign_fiber(j,k) only allowing ELG
9:         for each unassigned fiber k of this petal do
10:            Try assign_fiber(j,k) to only SS or SF
11:        if Number of SS ≤ 10 then
12:            Replace ELG (then LRG) to an SS until there are 10 SS
13:        if Number of SF ≤ 40 then
14:            Replace ELG (then LRG) to an SF until there are 40 SF
```

`improve` improves the plan applying improve_fiber to all of the fibers that concern it

`improve_from_kind (kind)` for every concerned petal, tries to assign unassigned fibers to SS or SF, to release an other fiber (formerly assigned to a SS or SF) that would be reassigned to a regular galaxy (kind is SS or SF when we call it). There is also the function `improve_fiber_from_kind` that does the same but on only a fiber.

Algorithm 5 Improve_from_kind (kind,j0,n)

```
for each plate j from j0 to j0+n do
    for each petal p of this plate, in a random order do
        for each unassigned fiber k do
            initialize a set of variables kp, g, gp
            for each available galaxies g of k do
                for each fiber kp of p assigned to a galaxy of kind do
                    if no conflict & g is worthier than previous one then
                        memorize kp, g, gp
            if g ≠ −1 then
                Reassign kp to g
                Assign k to a galaxy gp of kind kind
```

`update_plan_from_one_obs` updates the plan formerly made so that if for example on the current minus *Analysis* we observed a QSO which the analysis reveals fake but is still planed to be observed again, we will unassign the corresponding fiber and try to reassign it with improve_fiber and improve_fiber_from_kind

Algorithm 6 Update_plan_from_one_observation(j0,end_plan)

```
get the list of galaxies observed by this plate that are discovered fake or target
for each of those galaxies g do
    get the list of tile-fibers that are supposed to observed this galaxy further in the plan j0 ≤ jp ≤ end
    for each of those tile-fibers (jp,kp) do
        Unassign (jp, kp) ⟷ g
        run improve_fiber(j0+1,end-j0,jp,kp)
        if jp, kp hasn't been assigned then
            run improve_fiber_from_kind(SF,j0+1,end-j0,jp,kp)
        if jp, kp hasn't been assigned then
            run improve_fiber_from_kind(SS,j0+1,end-j0,jp,kp)
```

6.6 Displaying results

`results_on_inputs` displays some statistics on input files (recall input features, print statistics on the number of fibers with 0 galaxies within reach, 1 galaxy within reach etc. out to 19 galaxies within reach ; number of available tile-fibers for a galaxy, ...)

`display_results` writes some statistics and provides the tex-formatted results to make Table **??**

`print_free_fibers` print histograms on free fibers towards/regarding petals, for everything, for only SS and for only SF

7 Rules of assignment

— Of course, a tile-fiber is only assigned once
— two observations of a galaxy can't be separated by less than `InterPlate` (200) plates
— two observed galaxies can't be separated by less than `Collide` (set by find_collision)
— in last pass, only ELG, SS and SF are considered
— there can't be more than 10 fibers assigned to SS on a petal
— there can't be more than 40 fibers assigned to SF on a petal
— basically, we choose best galaxy from at the same time available, observed less than maxgoal(kind) times and unassigned ones. Among them, we take those with highest priority, and then the one which has been seen the least number of times
— between an already observed Ly-α and an unknown QSO, one chooses the already observed Ly-α

In nature we only know the basic type (QSO, LRG, ELG, SS, SF) of the galaxies of the catalog. The precise type is only infered by a simulation coded in python, but is generated with randomness. So in the algorithms, we can know the nature of the object only when it has been observed once at least. There are two way of assignment : either observe tile after tile and foresee the assignment of the tile just before observing it ; otherwise, we can make a plan for a definite

number of succeeding tiles. In such a plan, we try to optimize assignment, but of course, we can't know the precise type (fake, ...) so it is possible to foresee observing for example 4 times a fake QSO. We won't do it, so after having planned, when we begin to apply it and do the real observation, if we have just previously observed a fake QSO, we will remove further observations in this plan. We then try to assign the released tile-fibers. In the code, a difficulty was optimization at this point : we could have replanned everything in the plan after an update of information, but we can't because that would take too much time. So we only try to reassign released tile-fibers.

The idea is to get most information possible in the first pass, assigning independently tile after tile, and then, after this first pass, make a plan for everything else, when we have information on most of the QSO and LRG (the only kinds we want to observe several times). The plan will be better that way, because we do less mistakes than if we would do it from the begining.

8 Algorithm

`main` proceeds by loading input files : features of galaxies, reading in the galaxies catalog to make G, the positions of the fibers on the plate to make pp, the plate centers and positioner locations to make P.

We then collect available galaxies for each fiber for each plate, and compute the inverse map (available fibers for each galaxies, useful for improvement functions).

We next, tile after tile, do the assignment of galaxies. We have tried several strategies. The strategy that produced the best results, which will be the strategy of reference, consists of making only one plan, then applying it. We first run new_assign, then 3 rounds of improvement : improve, redistribute, improve, redistribute, improve.

Algorithm 7 Assignment of reference in main program

Phase I - Make a plan for all plates

1: Run, globally, on the list plates from 0 to last :
2: "New" assign fibers
3: Improve
4: Redistribute
5: Improve
6: Redistribute
7: Improve

Phase II - Apply the plan

1: **for** each plate j of the plan, in order **do**
2: Possibly "pyplot" the planned observed tile
3: Real observation is here
4: Update information collected on the previously observed Analysis'th past tile with update_plan_from_one_obs

A standard output display of an execution look like that (for assignment plan and applying) :

```
# Read 71,998,144 galaxies from /project/projectdirs/desi/mocks/preliminary/objects_ss_sf0.rdzipn
# Read 10,666 plate centers from /project/projectdirs/desi/software/edison/desimodel/0.3.1/data/footprint/desi-tiles.par and 5000 fibers from /project/projectdirs/desi/software/edison/desimodel/0.3.1/data/focalplane/fiber-pos.txt
# Start building HTM tree at 13.8 s
# ... took : 25.5 s
# Begin collecting available galaxies
# ... took : 31.6 s
# Begin computing available tilefibers
# ... took : 1 mn 12.5 s
# Start assignment at : 2 mn 27 s
# Begin new assignment :
50,518,743 assignments on all left next plates
# ... took : 18 mn 38.3 s
# Begin improve :
565,801 more assignments (1.120 % improvement)
# ... took : 36.5 s
```

```
# Begin redistribute TF :
1,760,465 redistributions of couples of TF
# ... took : 23.3 s
# Begin improve :
206,948 more assignments (0.405 % improvement)
# ... took : 30.2 s
# Begin redistribute TF :
1,575,337 redistributions of couples of TF
# ... took : 23.1 s
# Begin improve :
92,508 more assignments (0.180 % improvement)
# ... took : 26 s
# Begin real time assignment at 23 mn 24.5 s
- Plate 0 : 550 not as - 3852 unas & 2772 replaced
- Plate 1 : 98 not as - 3784 unas & 2773 replaced
- Plate 2 : 104 not as - 3985 unas & 2844 replaced
- Plate 3 : 121 not as - 3602 unas & 2563 replaced
- Plate 4 : 105 not as - 3773 unas & 2685 replaced
- Plate 5 : 85 not as - 3797 unas & 2853 replaced
- Plate 6 : 145 not as - 3537 unas & 2514 replaced
- Plate 7 : 92 not as - 3964 unas & 2856 replaced
- Plate 8 : 123 not as - 3714 unas & 2578 replaced
- Plate 9 : 143 not as - 3543 unas & 2378 replaced
- Plate 10 : 148 not as - 3528 unas & 2472 replaced
- Plate 11 : 86 not as - 3507 unas & 2885 replaced
```

9 Results

9.1 Results on the input catalog

Here are some statistics on the input galaxies catalog :

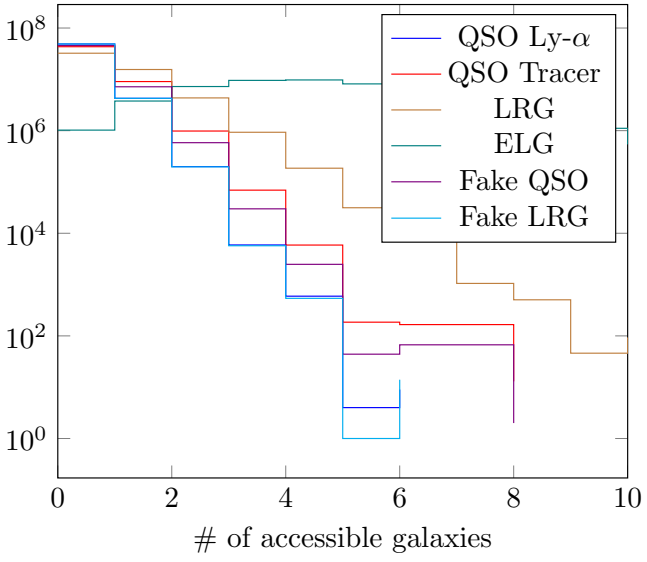


FIGURE 4 – Available galaxies (by kind) for a single fiber on a single tile

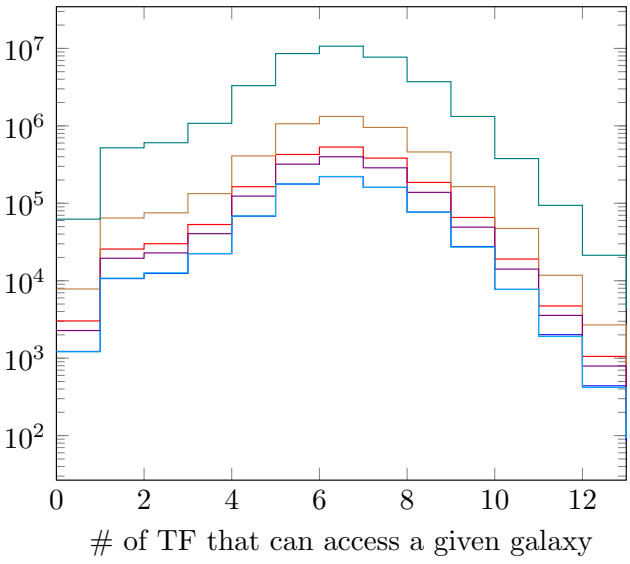


FIGURE 5 – Available tile-fibers for a galaxy (by kind)

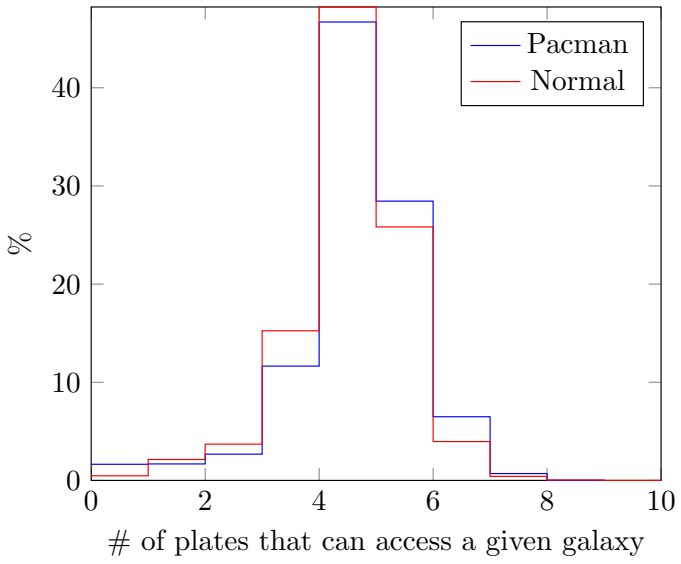


FIGURE 6 – Available plates for a galaxy (without 5th pass)

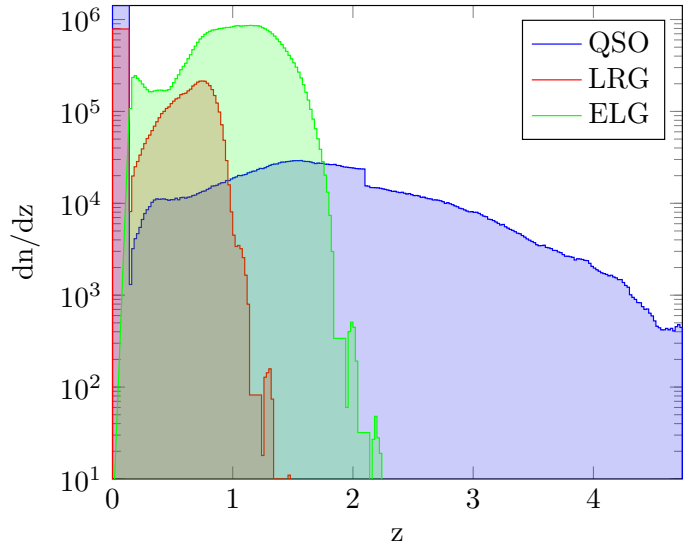


FIGURE 7 – Distribution of objects as a function of redshift

9.2 Results of the assignment

We run the program with all information (prior knowledge of information on fake, target, etc) to compare with realistic simulation, this result is interesting.

The results are then written using `display_results`, and each assignment/improvement function display its own statistics.

We precise that we have indeed exactly 10 SS and 40 SF per petal. We have taken a reference strategy, which was a trade-off between computation time and quality of results. We use improvement functions in a way such that they are still efficient given the time they take (because the more improvement execution we launch, the less they are efficent). And from this reference strategy, we change parameters to see the effects of them on the program. We also compare to other strategies.

The best strategy (discovered until now) is the one where we first look at 2000 tiles, tile after tile runing simple_assign on each tile without plan, and we then do a plan for all left tiles, before improving it only with an SF-improve.

In the second plan, we can experimentally see that after a first SF-improvement ($+ \sim 3.6\%$), even if we redistribute, a second one is almost useless ($+ \sim 0.1\%$), whereas a second simple-improve is still efficient after a redistribution ($+ \sim 0.4\%$), but useless without a redistribution step ($+ \sim 0.01\%$). An other kind of kind-improvement (instead of SF and SS ($+ \sim 0.5\%$)) is also almost useless. Furthermore, the computation time is far bigger for SF-improvement (5 mn) than for other.

The most results-sensible feature in it is the number of plan/applying and their plates sets. One could try to find a better one.

9.3 Sum-up of assigned galaxies

In the reference strategy, we have 50,095,513 assignments in total (93.54% of all fibers). The weighted score of a certain kind is defined as : $Score(kind) = 100 \cdot \frac{\sum_{g \in kind} obs(g)}{\sum_{g \in kind} goal}$ where $obs(g)$ is the number of times g is observed.

A sum-up table of general results on assignment is provided on Table 3.

	Times observed										
	0	1	2	3	4	5	Total	Fiber used	Once observed	observed %	weighted %
QSO Ly-α	0	2	5	15	25	1	49	166	49	98.468	66.747
QSO Tracer	1	118	0	0	0	0	119	118	118	98.487	98.487
LRG	16	38	243	0	0	0	298	524	281	94.352	87.854
ELG	540	1,870	0	0	0	0	2,411	1,870	1,870	77.572	77.572
Fake QSO	1	88	0	0	0	0	90	88	88	98.472	98.472
Fake LRG	2	47	0	0	0	0	50	47	47	95.061	95.061
SS	72	67	0	0	0	0	140	67	67	48.252	48.252
SF	1,129	270	0	0	0	0	1,400	270	270	19.300	19.300

TABLE 3 – Densities (objects/deg^2) as as a function of # of observations (with total), and % observed, once and weighted

9.4 Pacman plate

On the Table 4, we present results of the assignment taking for input the fiber locations files of the pacman (plate deprived of 4 petals) used in case of run out of money. Every parameter is the same. As one can notice, they are quite similar to regular ones.

	Times observed										
	0	1	2	3	4	5	Total	Fiber used	Once observed	observed %	weighted %
QSO Ly-a	1	1	4	14	27	0	49	166	48	97.593	66.818
QSO Tracer	2	117	0	0	0	0	119	117	117	97.575	97.575
LRG	17	33	247	0	0	0	298	528	280	93.987	88.411
ELG	493	1,918	0	0	0	0	2,411	1,918	1,918	79.540	79.540
Fake QSO	2	87	0	0	0	0	90	87	87	97.579	97.579
Fake LRG	2	47	0	0	0	0	50	47	47	94.608	94.608
SS	70	69	0	0	0	0	140	69	69	49.576	49.576
SF	1,122	277	0	0	0	0	1,400	277	277	19.830	19.830

TABLE 4 – Same than Table 3 but with the pacman

Furthermore, there are 50,938,132 assignments in total (92.9358 % of all fibers).

9.5 Free fibers

Here are the histogram of petals as a function of free fibers on Figure 8 and the number of free fibers for each plate, in inscreasing order on Figure 9.

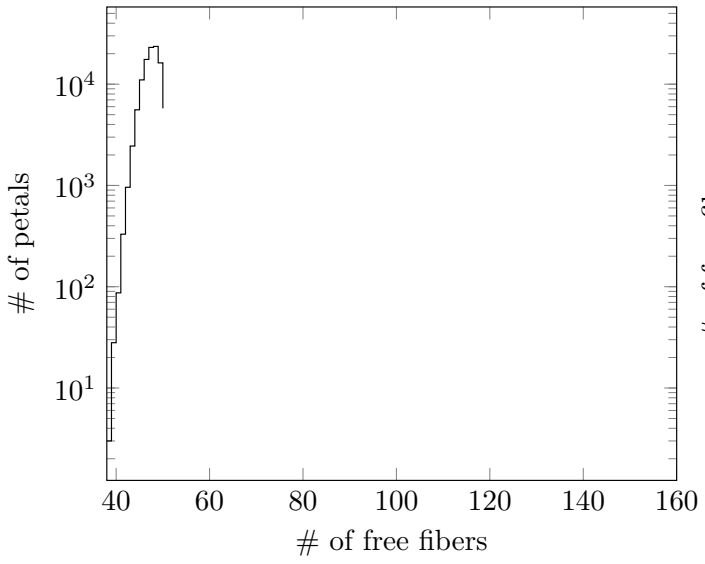


FIGURE 8 – # of petals with that many free fibers

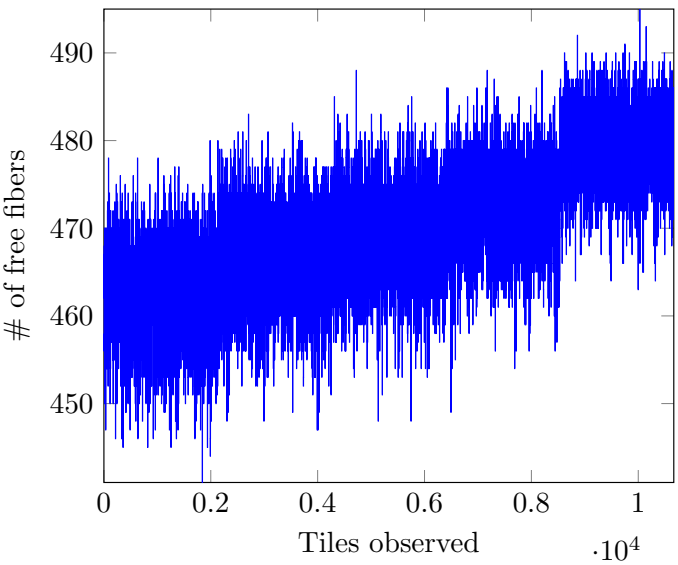


FIGURE 9 – Free fibers as a function of time (plates)

On the Figure 10, one can seen the proportion of observed objects as a function of their density. "usq" stands for unity of square degrees, its the area of the sky reachable by a single fiber. Keep Figure 4 in mind while reading this one.

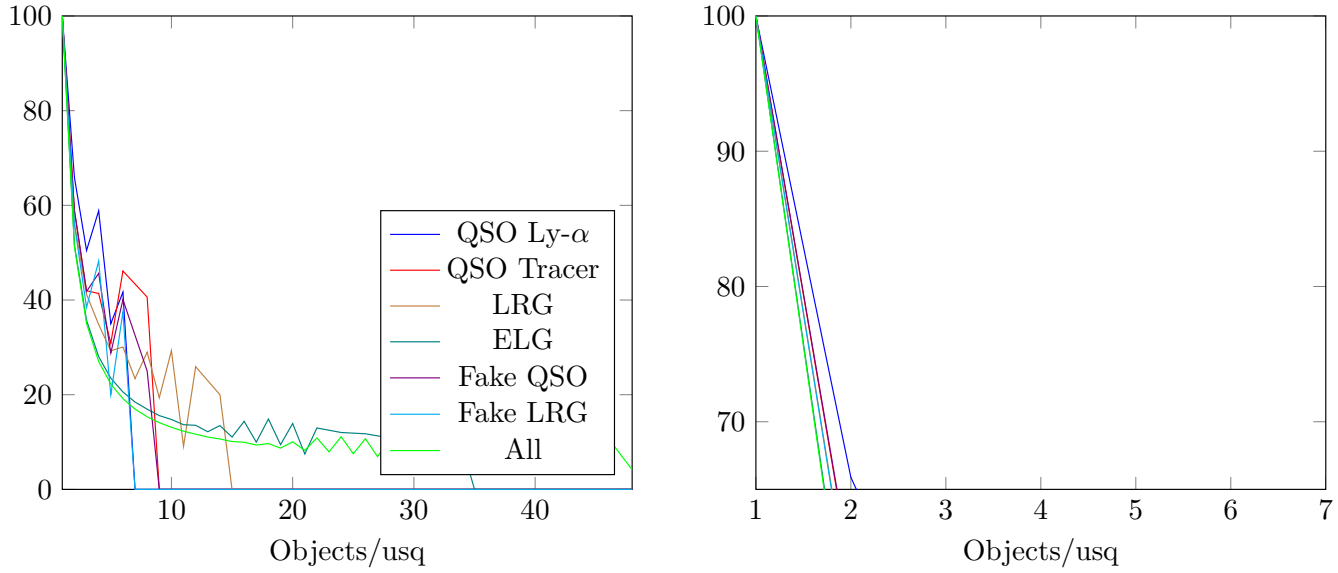


FIGURE 10 – % of observed galaxies as a function of objects density, wide and zoom

9.6 Redistribution-Improvement step

We see, in the second plan, what is the effect of several redistribution-improvement executions. Here is presented, in the Table 5, the number of redistributions, number of additionnal assignments, with the improvement in % and the total time taken for until this step.

Step	# red	# +as	+ %	Time
1	2,204,977	167,741	.418	40s
2	2,060,851	76,509	.190	1mn 20s
3	1,965,994	19,273	0.048	2mn
4	1,945,418	10,738	0.027	2mn 40s
5	1,929,917	7,999	0.020	3mn 20s
6	1,921,331	5,745	0.014	4mn

TABLE 5 – Results of several redistribution-improvement steps

9.7 Results on Ly-α particularly

The figure 11 gives the histogram of proportion of assigned Ly-α and the number of observations as a function of available TF.

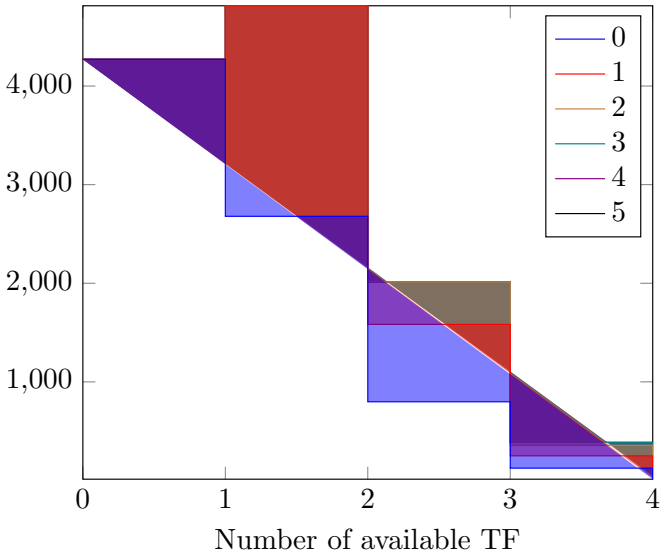


FIGURE 11 – # of QSO Ly-α (with their number of observation) as a function of available tile-fibers

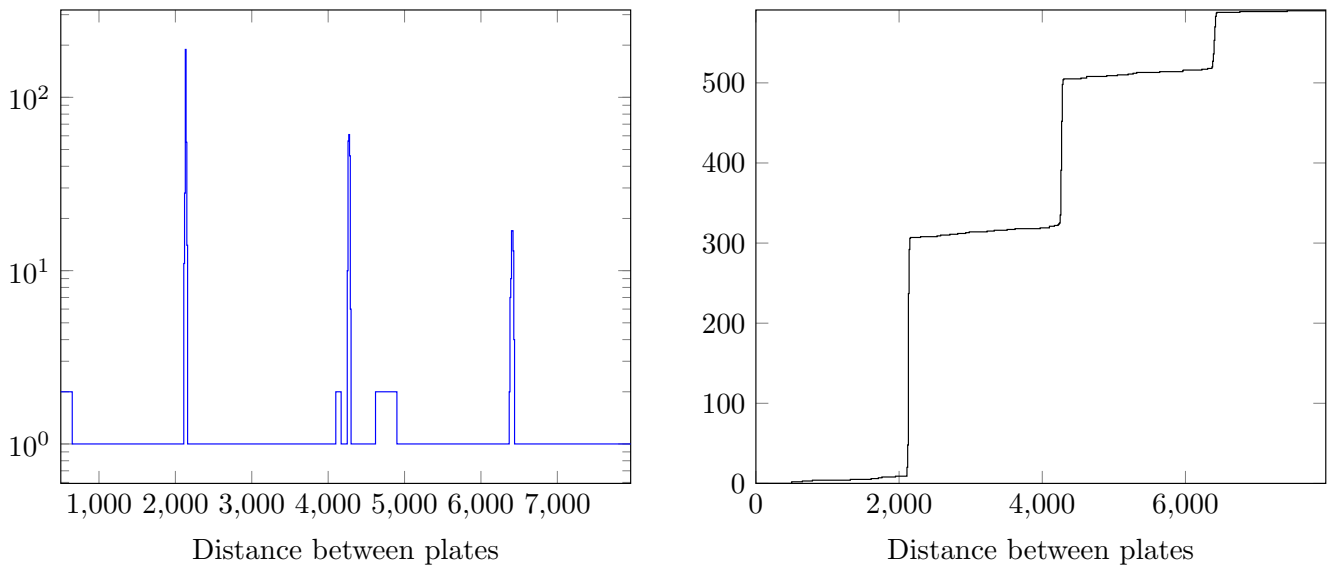


FIGURE 12 – Histogram of distances (in number of plates) between two consecutive observed QSO Ly-α and integral

9.8 Study on evolution over time

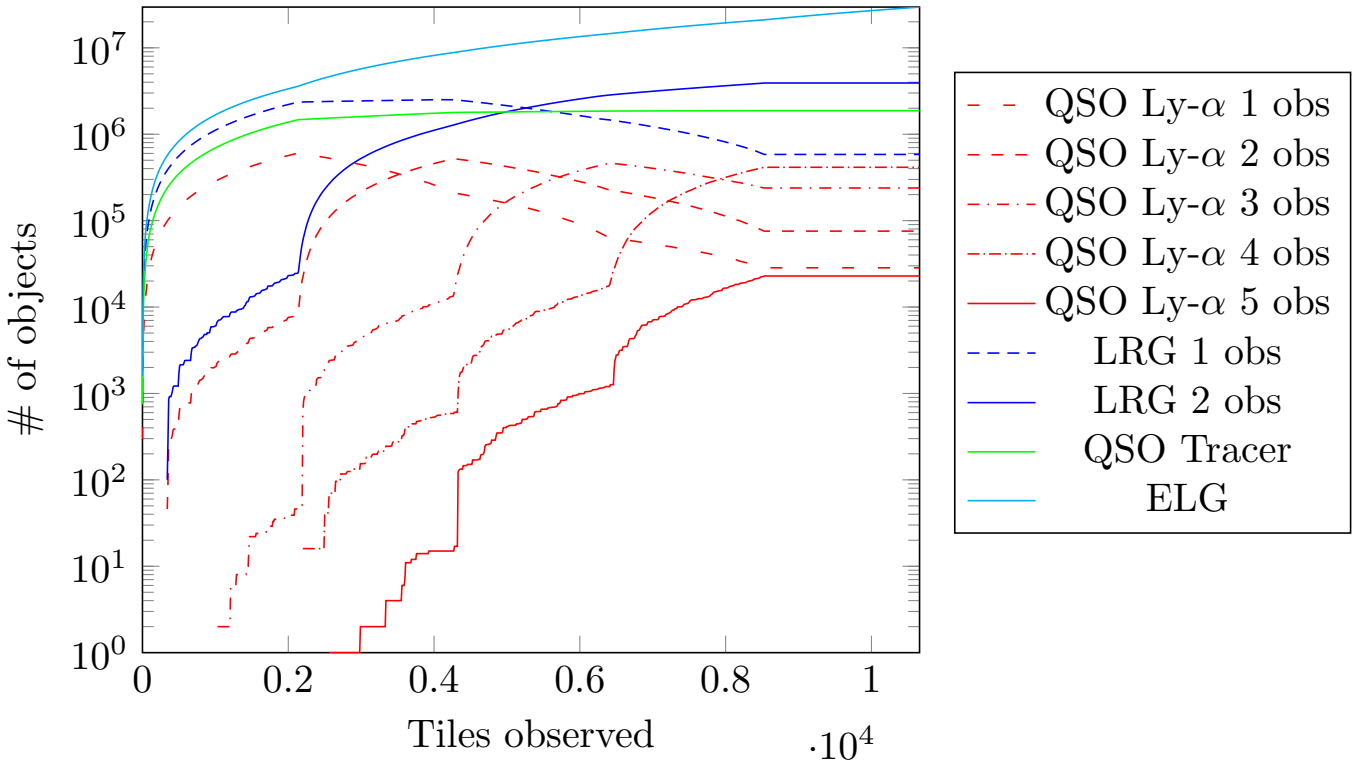


FIGURE 13 – Observed galaxy kind as a function of time (plates seen)

9.9 Shifting parameters

In the following, we change a little bit parameters around the references values to observe their influences on results.

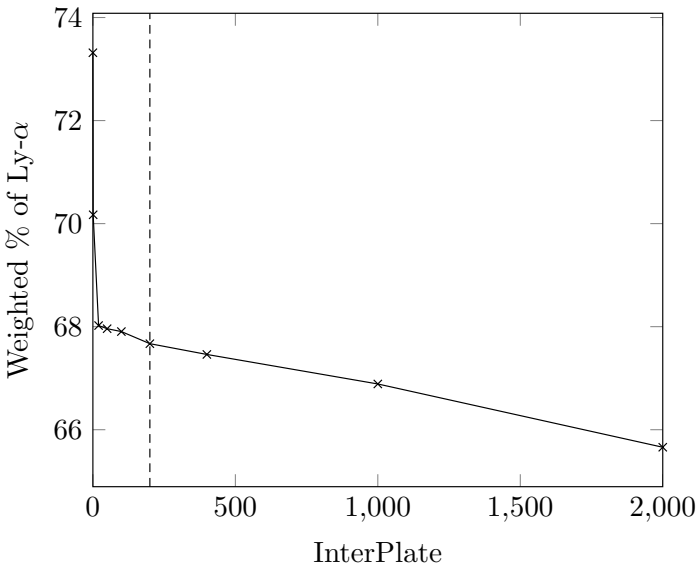


FIGURE 14 – % of Ly-α observed as a function of required minimum interval between plates observing the Ly-α

10 Collision problem

There is an option in the features file that indicates whether we choose the exact geometry of the fiber positioners. If not, collisions are computed with only circles at the position of the fiber holder, of radius *AvCollide*, which we calibrate such that we get the same collision rate than with the exact geometry ($\sim$ 10.5% when we take all plates). If the exact geometry is on, the modules collision.h and collision.cpp are used to check. They then build sets of circles or segments, and check collisions between those belonging to the first fiber positioner to the ones belonging to the second one. Computationnally cheap methods were used for segment-segment and circle-segment checking, because checking functions are called a lot of time, and optimization of those methods lead to considerably decrease total computation time. That is why we use circle and segments, not only segments : creating an arc with segments needs a lot of them, but only one circle.

Figure 15 is the histogram of distances between two galaxies, in a collision case, and its integral.

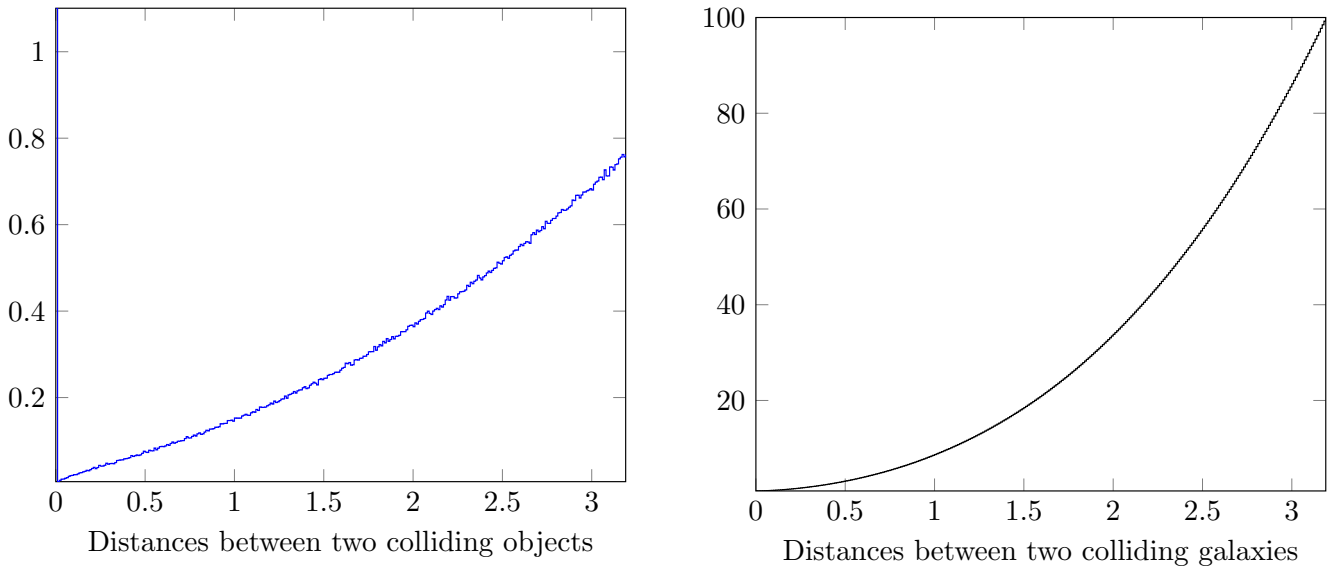


FIGURE 15 – Histogram of distances between galaxies in collision case

11 Tile ploting

The function pyplotTile builds a tile.py file, which can be executed manually with python tile.py to create the pdf plot. Only objects that would need at least one further observation are projected on the plate and ploted. The function has to be called just before real observation time, otherwise, if it is called at the end for example, information on number of observations of an object would be the one that we have at the end, and not on the way. A black dot is ploted when there is an unassigned fiber at some place, not to mix up with fiducials (totally white), places where there is physically no fiber. Watching at a dozen of tile plots, one can convince hiself that the assignment is very close to the optimum. Notice that on plot, one can for example see a fiber looking at an ELG though it could observe an LRG instead. It comes from the improvement functions, that "anneal" the assignment, and then the program plans anyway to look at the LRG but later, which leads to optimize the assignment.

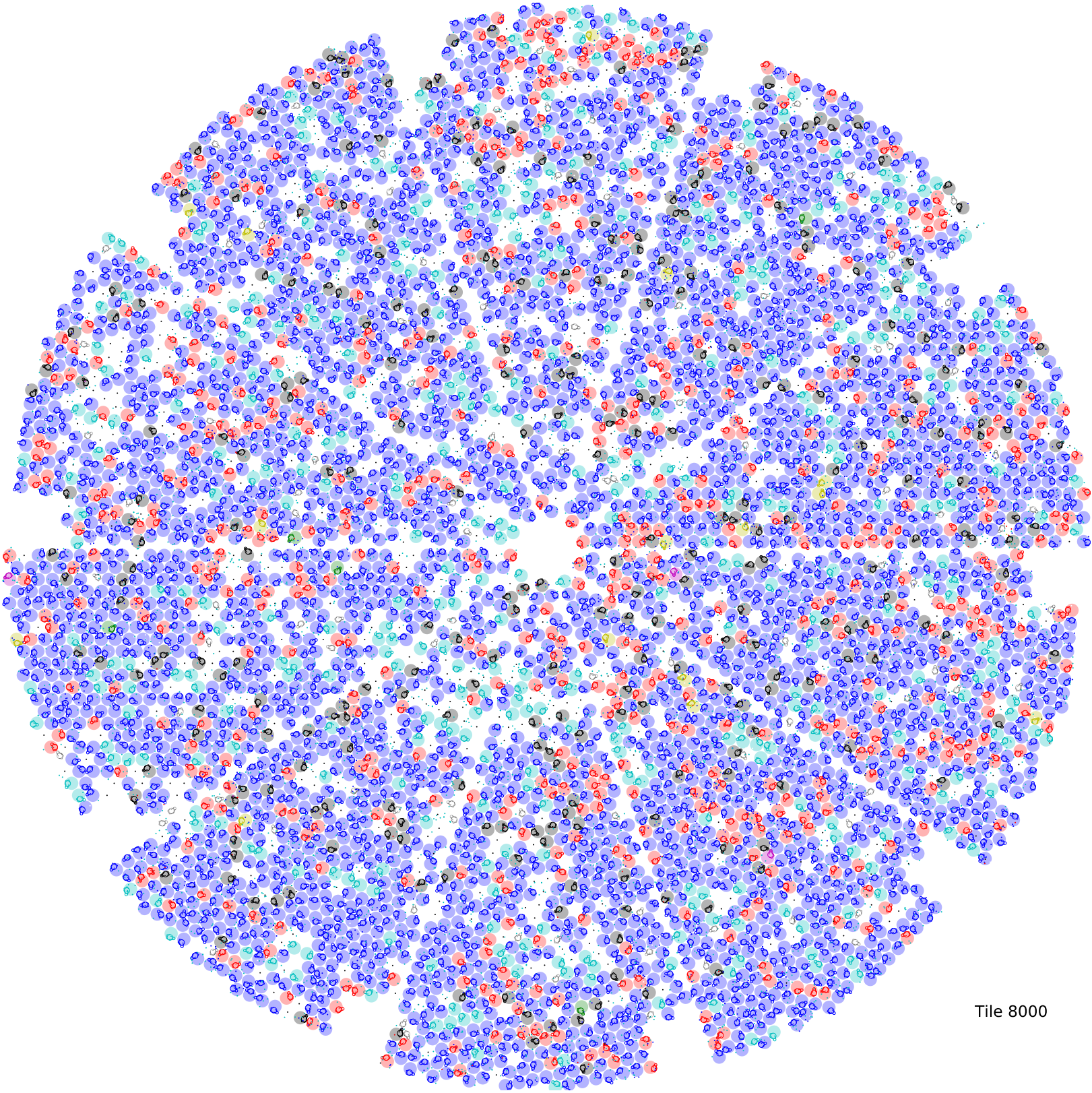


FIGURE 16 – Tile

12 Possible improvements

We havn't tried all possible strategies, so it is possible that an other combination of runing of the functions could lead to better results. One could also try to search for an "orthogonal" (of the 3 other ones) improvement function. To make plans, we could use an automaton, which would be automatically able to use all kinds of assigment/improvement functions, and would go through all fibers at random to assign/improve randomly, and the controlable parameter would only be the time we want it to work. Nevertheless, a lot of efforts now lead to little improvements of the results. It is likely that we are almost at the global optimum. We only have a doubt on the number of plans/applying and their sets of plates that could improve significantly.

The problem of colliding fiber positioners is addressed in a simplified way (circles around a fiber) which can be precised.

We can adapt a little bit the code in order to do assignment and improvements without SS and SF, and only add SS and SF just before launching the observation, using the "replace" function in global.cpp. It's not likely to improve a lot because improve-SS and SF must already improve this way.

The update function might still be improved to lead to more reassignments, and an improvement execution on the current plan could be efficient after several hundred updatings, since some "redistribution" happened with updatings.

We could make plans with only Ly-α assigned, launching improve function, and then adding separately LRG then improving, and adding again ELG and improving. It's likely that it won't increase results because as we saw before, doing assignments separately in new_assign already didn't improved results. This kind of optimization must actually be already done by selecting with priorities and executing improvement functions.

Make a plan without SS and SF (to keep degrees of freedom for improvement functions) and then assign them only just before the observation.

Do a function which puts all observations by the first tilefiber possible, after redistribution-improvement.

13 Effect on correlation function

One of the difficulties is that we have not to bring experimental bias the way we do the assignment. The fact that a lot of rules are used in the assignment make it more likely that a bias is created.

Références