

Documentation of the C code used to calculate the path persistent homology (aka pph) up to diagrams of dimension 0 and 1

Rafael Polli Carneiro

Contents

1	Initial Observations	1
2	How The Program is structured	1
3	Headers	1
3.1	<i>definitions.h</i>	2
3.1.1	Data types	2
3.1.2	Functions	3
3.2	<i>basis_of_vector_space.h</i>	4
3.2.1	Data Types	4
3.2.2	Functions	5

1 Initial Observations

Be warned that henceforth we will admit the following conventions:

- i) Whenever talking about vector spaces, namely V , we have that the field acting onto V is going to be \mathbb{Z}_2 ;
- ii) Every vector space here has finite dimension;

2 How The Program is structured

Everything here, concerning the C code, is splitted into the two folders: *headers*, *src*. Inside headers one can find the data types which are used to perform the main algorithm, as well the respectively functions that operate with these data types. Now, *src* is merely the C implementation of such functions, or methods if someone prefers to call them. The C code called *main.c*, runs the algorithm desired to be calculated.

3 Headers

The folder Headers has the following tree structure:

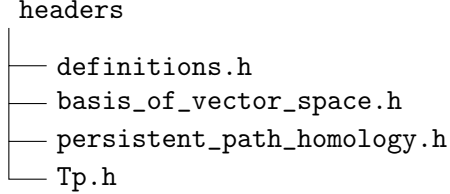


Figure 1: tree headers.

Now let's describe each file contained at **headers**.

3.1 *definitions.h*

Here, basic, yet fundamental data types are defined. To begin with, we have the following constants:

- **TRUE** or **FALSE**;
- **MARKED** or **NOT_MARKED**. Here, these flags address whether or not an element of a vector space, more precisely, an element of its basis, is marked or not.
- **EMPTY** or **NOT_EMPTY**. We say that the i th element of an array can be empty or not, meaning that nothing has been allocated at its memory space, if it is empty, and the contrary otherwise;
- **SORTED** or **NOT_SORTED**. ???;

3.1.1 Data types

The data types are:

- **boolean** = **TRUE** or **FALSE**;
- **vertex_index** = $0, 1, 2, \dots, N$. Given a graph (G, V) , each vertex (or node if you prefer to call it), will be enumerated by integers ≥ 0 ;
- **base_index**. Let V be a vector space. Then, considering $\beta \subseteq V$ as its base, with $\beta = \{v_0, v_1, \dots, v_N\}$, we say that $0, 1, 2, \dots$ will be elements of the **base_index**;
- **vector_index**. Any element of a vector space V , i.e., a vector, can be written as coordinates, given a fixed basis. Thus, let's say that $V \ni u = (0, 1, 1, 0, 1)$. Then the non-negative integers $0, 1, 2, 3, 4$ are

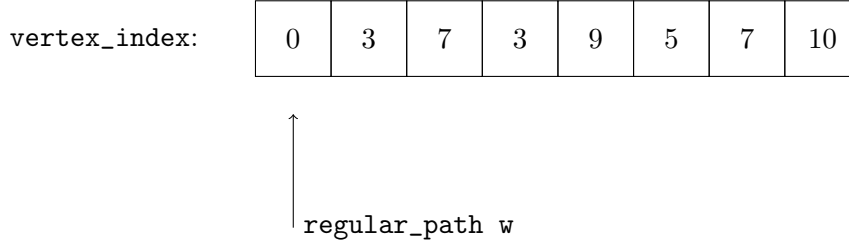


Figure 2: Representation of a regular path $w = [0, 3, 7, 3, 9, 5, 7, 10]$.

elements of the data type `vector_index`, with $u_0 = 0, u_1 = 1, u_2 = 1, u_3 = 0, u_4 = 1$;

- **dim_path.** Let (G, A) be a graph. Then $w = [a_0, a_1, \dots, a_N]$ is called a regular path in G if $\forall i \in \{0, 1, \dots, N\}$ the following holds

$a_i \in G$ and $(a_0, a_1), (a_1, a_2), \dots, (a_{N-1}, a_N)$ are edges of the graph.

The regular path w , in this case, is said to have dimension N . Consequently, `dim_path` will be the data type responsible of representing all dimension of possible regular paths being used in the future;

- **dim_vector_space.** The meaning of this data type can be understood immediately;
- **regular_path.** A regular path is exactly what was described up abpve, that is, $w = [a_0, a_1, \dots, a_N]$ is a regular path when each a_i is a node, for all i , and every pair $(a_0, a_1), (a_1, a_2), \dots, (a_{N-1}, a_N)$ is an edge. It is natural to represent a regular path as an array and, in fact, that is the way they will be represented in here. Thus, the type `regular_path` is, as expected, a pointer to `vertex_index` types. See Figure 2 for a visual explanation.
- **vector.** Considering V a vector space with basis $\beta \subseteq V$, then any element of V can be seen as its coordinates. For instance, let $w = (0, 0, 0, 0, 1, 1, 0) \in V$. The data type `vector` is, as `regular_path`, a pointer to `boolean`, which are elements of \mathbb{Z}_2 . **For large graphs, and due to the fact that the array vector is filled with many zeros (in my analysis), this data type is not the best to be used. I WILL CHANGE IT FOR A LIST!!!**

3.1.2 Functions

Finally, the following functions, down below, will implement some routines over the data types defined into `definitions.h`. They are:

are_these_regular_paths_the_same

Parameters : regular_path, regular_path, dim_path.

Objective : Check if two regular paths w_1, w_2 are equal. In case positive it returns TRUE, and FALSE otherwise.

Return : boolean.

is_this_path_a_regular_path

Parameters : regular_path, dim_path.

Objective : It checks if an the array $w = [a_0, a_1, \dots, a_N]$ is a regular path, that is, the pairs $(a_0, a_1), (a_1, a_2), \dots, (a_{N-1}, a_N)$ are, indeed, edges of the graph. This function is important specially when taking a border operator of a regular path which, in most cases, the outcome can incorporate non regular paths. It returns TRUE if w is a regular path, and FALSE otherwise.

Return : boolean.

is_this_vector_zero

Parameters : vector, dim_vector_space.

Objective : It checks if a vector v of a vector space is zero, returning TRUE if positive and FALSE otherwise

Return : boolean.

sum_these_vectors

Parameters : vector, vector, dim_vector_space.

Objective : Let w_1, w_2 two elements of a vector space. Then this function returns a vector with the value equal to the sum $w_1 + w_2$.

Return : vector.

3.2 basis_of_vector_space.h

This header will provide us an abstraction of vector spaces which are spanned by a collection of regular paths. Bare in mind that in our context we have a sequence of sets of regular paths, indexed by $i \in \mathbb{R}_{\geq 0}$, given by

$$R_i^p = \{[a_0, a_1, \dots, a_p]; f(a_0, a_1) \leq i, f(a_1, a_2) \leq i, \dots, f(a_{p-1}, a_p)\},$$

where a_0, a_1, \dots, a_p nodes of a graph and $f : G \times G \rightarrow \mathbb{R}_{\geq 0}$ a matrix. This provide a sequence

$$\mathbb{R}_i \subseteq \mathbb{R}_j,$$

whenever $0 \leq i \leq j$. This is called a filtration, and it will provide us times which homological features appears and dissapear. With that in mind, for every regular path $w = [a_0, a_1, \dots, a_p]$, we define

$$\text{allowtime}(w) := \inf\{i \in \mathbb{R}_{\geq 0}; f(a_0, a_1) \leq i, f(a_1, a_2) \leq i, \dots, f(a_{p-1}, a_p)\}$$

and

$$\text{entrytime}(w) := \min\{\text{allowtime}(w), \text{allowtime}(\partial(w))\};$$

with f that matrix associated with the graph and ∂ the border operator.

3.2.1 Data Types

With these operators in mind we start by the following data types:

- **tuple_regular_path_double**. Given a base β of regular paths, then **tuple_regular_path_double** is a structure composed by the i th regular path of the base β and by the allow time of such regular path. See Figure 3.
- **base**. This is a structure that will be responsible to store all elements of a base which are regular paths of same dimension. Thus **base_matrix** is a pointer to **tuple_regular_path_double** where the regular paths must have the dimention given by **dimension_of_the_regular_path**. The dimension of the vector space spaned by the base **base_matrix** is stored at **dimension_of_the_vs_spanned_by_base**. Finally, **marks** is an array of the same size of **base_matrix** where it shows if the i th element of the base is marked or not. See Figure 3. Notice that in the Figure 3 only **base_matrix** is represented, while the other features are left inside the center dots (this is done beacuse **base_matrix** is the object that should be paid more atention).
- **collection_of_basis**. This is a structure which can be seen as the union of every base indexed by the dimension of the paths that generates each vector space. Here **max_of_basis** counts how many basis do we have, taking into consideration that we start counting from zero. See Figure 3.

3.2.2 Functions

Now the functions operating with these data types:

alloc_all_basis

Parameters : unsigned_int, unsigned_int, double**.

Objective : Given an integer N , this function will allocate $N + 1$ basis, taking as reference the amount of nodes of the graph and its network weight matrix.
(Obs: the function `generating_all_regular_paths_dim_p` is used)
Return : `collection_of_basis*`.

generating_all_regular_paths_dim_p

Parameters : `collection_of_basis*`, `dim_path`, `unsigned_int`, `double**`.

Objective : Given a `collection_of_basis*` this function will allocate the base of all regular paths of dimension p , using the properties of the network.

generating_all_regular_paths_dim_p will be used by the function above, in a loop throughout the path dimensions.

Return : `void`.

Now we need to describe how both functions work before we carry on.

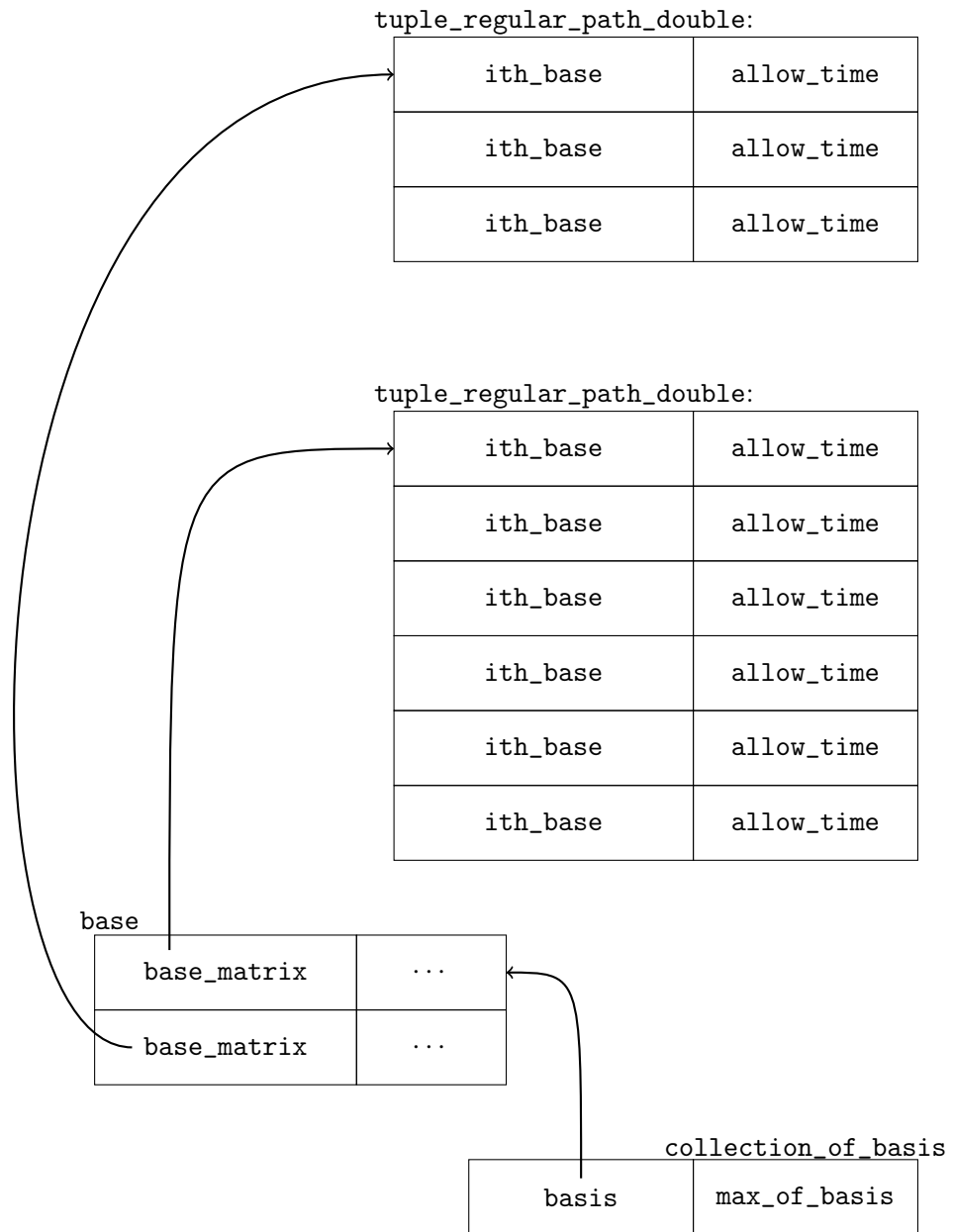


Figure 3: Visual representation of the data types defined at the file `basis_of_vector_space.h`.