Documentation: Simulated Annealing Algorithm (v. 101)

Fanor Balderrama

9/1/2021

Contents

Introduction **2**

Overview of the algorithm **2**
Decision tree diagram for the algorithm . . . 2
The `sa.fun` Function . . . 4
The temperature function . . . 5
The objective function (`func.ct.loc`) . . . 5

Summary of the script **6**
Paramaters (constants) . . . 6
Code chunk 1 (unnamed) . . . 6
Code chunk 2: setup . . . 7
Code chunk 3: extract 2016 marginal poercentages for each CT . . . 7
Code chunk 4 (unnamed) . . . 8
Code chunk 5: extracting LFS variables . . . 8
Chunk 6: saved_wrangled . . . 9
Chunk 7: sa func . . . 9
Chunk 8: cma func . . . 9
Chunk 9: sa fit . . . 9
Chunk 10: save SA fit . . . 9

Performance and future improvements **9**

Introduction

The following document presents an overview of the Simulated Annealing Algorithm and summarizes the script `SA-first_run.Rmd`, where the algorithm, in its current version (v. 101) is coded. The script was initially authored by Vid Bijelic (`vbije016@uottawa.ca`). It was handed to us by the author on June 21, 2021 and adapted to run in the ODEN computing environment by Fanor Balderrama (`fanor.balderrama@utoronto.ca`). The script is in the R Markdown (`.Rmd`) format. Although it was intended to create a Word document as an output, there is no content in this document. As with any R Markdown scripts, the R code is run in code chunks and a YAML header at the beginning of the script contains the metadata and parameters.

The LFS data set used in this script is from month of January 2020 and it was obtained from the Odesi data repository (http://odesi2.scholarsportal.info/webview/) in the Stata format (`.dta` file). The Census 2016 data set used is titled "Census metropolitan areas(CMAs), tracted census agglomerations (CAs) and census tracts (CTs)", and it was obtained from the Statistics Canada website (https://www12.statcan.gc.ca/census-recensement/2016/dp-pd/prof/details/download-telecharger/comp/page_dl-tc.cfm?Lang=E)

Overview of the algorithm

The purpose of the simulated annealing algorithm is the modeling the population of each of the census tracts (CTs) of the Census Metropolitan Area (CMA) of Toronto with data more up-to-date than what is currently available from the 2016 census. This modeling makes use of the survey sample of recent months (January 2021) of the Labour Force Survey (LFS) to impute the population of each CT in the census. The algorithm works by drawing a sample of observations from the LFS and making changes to the sample until the new sample is considered to match the census data for that specific CT (within a certain margin of error) on variables that are not considered to change significantly between 2016 and 2021. These variables, which are called *matching variables*, include the portion of the population within age categories (`Total15_24`, `Total25_44`, `Total45_64`, `Total65_Over`), marital status (`MarriedOrCL`), education (`WithPSE`), immigration status (`Recent_Immigrants`), economic family type (`CouplesWithChildren`, `CouplesWithoutChildren`, `LoneParent`, `NotInFamilies`), sex (`Female`, `Male`), employment status (`EmploymentRate`), and income (`EarnLessThan30` and `EarnMoreThan30`) [1]. The rest of the variables are allowed to change and account for the socioeconomic changes in the CT between 2016 and January 2020. They are called *matched* or *testing variables* and include variables concerned with the occupation and industry of employment (`OccupationHealth`, `IndustryHealth`, `IndustryNeither`, and `IndustryRetail`). The specifics of how all these variables are obtained from the raw LFS and census data sets are included in the summary of code chunks 3 and 5.

The algorithm is coded in the `sa.fun` function in the script and it makes use of two auxiliary functions: the *objective function* and the *temperature function*, which are explained in more detail below. The algorithm also uses a number of internal variables: *curSmpl*, *newSmpl*, and *bestSmpl* represent the current sample, new sample, and best sample from the LFS, respectively, which are compared to the census data and updated to search for the sample that best matches the census. The variables k and j count the number of temperature and annealing iterations the algorithm goes through while looking for the best sample match (see the *Parameters section*). The specifics of these two types of iterations are discussed in the *temperature function* section.

Decision tree diagram for the algorithm

Figure 1 summarizes the simulated annealing algorithm that is run for each CT in terms of the decisions and assignment of values to the internal variables.

[1] THERE IS A POSSIBLE ERROR HERE: Employment rate should be a *testing variable* in my opinion.

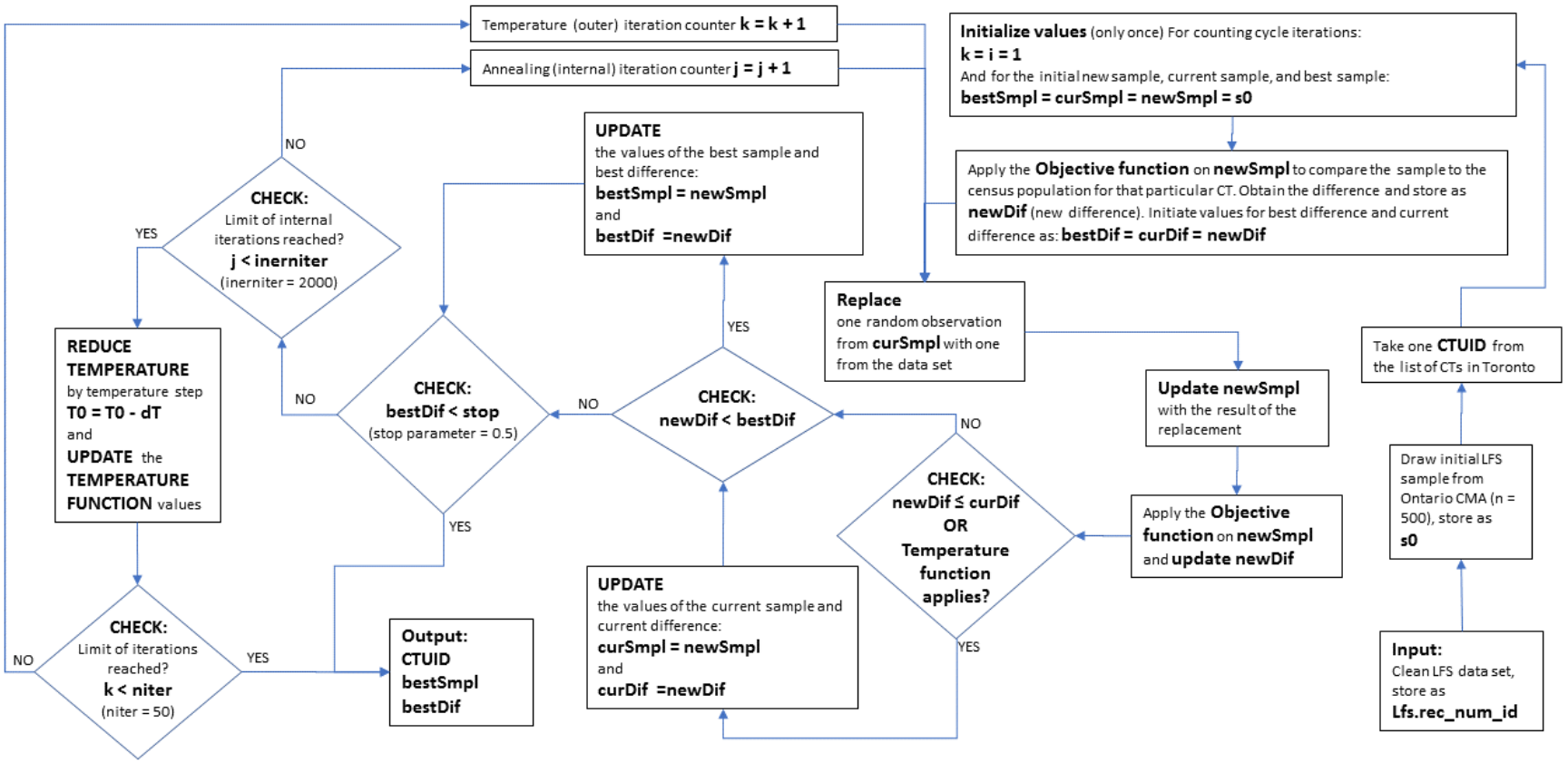

Figure 1: Decision tree diagram for the simmulated annealing algorithm

Data from the census for a specific CT is used as reference values. The algorithm starts when an initial sample of randomly chosen LFS observations is drawn, producing a current sample (*curSmpl*). The size of this sample is specified by the `lfssamplesize` parameter (see the *Summary of the script* section). The algorithm runs several consecutive cycles or iterations to modify this sample and check if the new modified sample (*newSmpl*) is closer to the census reference values. In each iteration the LFS sample is changed by replacing a randomly chosen observation with a randomly chosen observation from the entire LFS data set. If the new sample is closer to the census data, it becomes the best sample (*bestSmpl*) and replaces the current sample for the next iteration of replacement and comparison. The *objective function* is used to quantify the difference between the new sample of LFS observations and the census CT data. The new sample is considered to be better than the current sample if any of two criteria is met:

1. The *objective function* determines that the new LFS sample is closer to the CT reference data than the current sample, or
2. The temperature function, applied to the current and new samples, returns the value of `TRUE`

The algorithm stops running additional iterations when the best sample of LFS observations matches a census data according to any of these two criteria:

1. The objective function determines that the difference between the best LFS sample and the reference census data is less than the `stop` parameter.
2. The algorithm has gone through a maximum number of iterations, which is defined as a function of the maximum number of allowed maximum temperature and annealing iterations (see *Parameters* section) and the first criteria has not been met yet

At the end of the last iteration, the function returns the best sample and the id number for the CT whose population has been imputed. The sample is in the form of the record IDs of the LFS observations in the sample. The algorithm can be run again to impute the population of the next CT.

The `sa.fun` Function

This function specifies the operations of the simulated annealing algorithm. It is called only once, in chunk 9 of the script. It uses two auxiliary functions: the *objective function*, which is specified in chunk 8 and passed to this function as an argument; and the *temperature function*, which is defined inside the code for the `sa.fun` function and is used for one of the criteria that determines when a new sample is better than the current sample in consideration.

Arguments

The `sa.fun` function requires the following arguments:

- **func** : A function. When this function is called in chunk 9, the objective function from chunk 8 is passed.
- **s0** : This is the initial random sample, in the form of a vector of record numbers from the LFS "REC_NUM" variable
- **smpl** : The LFS sample size. A sample size of 500 is used in this version of the algorithm and specified among the parameters.
- **niter** : The number of outer or "temperature" iteration; 50 in this version of the algorithm. It is specified among the parameters.
- **inerniter** : The number of inner or "annealing" iterations; 500 in this version of the algorithm. Specified among the parameters.
- **T0** : Initial temperature, used in the temperature function. The value given to this argument when called is 0.01; specified among the parameters.
- **dT** : Change in temperature to cool down the initial temperature in the temperature function. The value given to this argument when called is 0.0002; specified among the parameters.

- **stop** : Argument passed for the stop parameter in the algorithm, currently given the value of 0.5; specified among the parameters at the beginning of the script.
- **temp.stop** : A numeric value, currently assigned the value of zero, specified among the parameters [2]. .
- **rec__num__id** : A numeric vector for the set of LFS record numbers. Currently assigned the entire "REC_NUM" variable of a clean LFS data set.
- **lfs** : A data set from where the samples are drawn for the algorithm. Currently the entire clean LFS data set (for January 2021) is assigned.
- **cen.ct** : A census data set for a specific CT (census tract). Samples of LFS observations will be sampled iteratively to match these census reference data values.
- **ctuid** : A number identifying the CT for which a synthetic population is being modeled.

The temperature function

The *temperature function* is an auxiliary function used inside the simulated annealing algorithm in the `sa.fun` function. It is mathematically defined as:

$$i < e^{\frac{curDif - newDif}{T0}}$$

Where:

- i is a random number between 0 and 1 drawn every time the function is called
- $curDif$ and $newDif$ are the current and new differences (between the current and new samples and the census CT data), respectively, produced by the objective function
- $T0$ is a temperature parameter originally specified as .01, but lowered at the end of each temperature iteration.

The function returns `TRUE` if i is less than the expression on the right hand side or `FALSE` otherwise. The output of this function is used as one of the criteria that determines if a new sample is better than the current sample.

This function is based on the Metropolis-Hasting algorithm. If a better sample cannot be found after running a maximum number of annealing iterations (i.e., when the iteration counter j equals the parameter `interniter`), then the criteria for finding a better sample is relaxed by reducing the value of $T0$ ("reduce temperature" in figure 1) by the value of the parameter `dT` and trying again with a new turn of annealing iterations (i.e., the iteration counter j resets). This relaxation of the criteria will be performed a maximum number of times (specified by the parameter `niter` and counted by the variable k), after which the algorithm ceases and the best LFS sample is returned. The cycles before and after the relaxation of the criteria are called temperature iterations or temperature loops. The iterations happening inside each temperature iteration are called annealing iterations or annealing loops [3].

The objective function (`func.ct.loc`)

The *objective function* gets the LFS data for the sample to be compared. The values for the variable of interest are summed for the sample and compared to the same sum of values in the census CT data. The difference between these two data sets is normalized over the sum of statistical weights (FINALWT) in the LFS sample. It must be specified that the LFS samples have 500 observations (LFS records), but each record has statistical weights that estimate the number of individuals expected to match the same characteristics in the population [4].

[2] CURRENTLY UNUSED IN THE ALGORITHM

[3] I would like to check if the temperature function works correctly and how often it returns `TRUE`. It might not be working as intended

[4] I STILL NEED TO CHECK THAT THE DIMENSIONS OF THIS COMPARISON MATCH

Arguments

The *objective function* takes in the following arguments, which must be specified:

- **lfsSample** : A numeric vector for the LFS observations (by record ID number) of a sample to be compared to the census CT data
- **lfs.cma** : A clean LFS data set.
- **lfs.cen.ct** : A clean census data set for a specific CT

Summary of the script

Paramaters (constants)

These parameters are defined in the YAML header. Most of these parameters are used in the algorithm, but some are currently unused. These parameters can be called in the R code with the prefix `params$`, e.g., `params$cma` .

- **rdsversion** : is is a record of the version of the algorithm under development. Currently 101, only internal.
- **lfsdataset** : the location for the LFS data set. In this case, only the January 2021 LFS data set was used.
- **census2016dataset** : the location for the census data set.
- **cma** : refers to the coding in the LFS for the CMA to focus on. The current value is 4, which is the variable code for the CMA of Toronto in the LFS. See the LFS documentation.
- **cmacode** : text value that denotes a specific CMA in the census data set. Used in the filtering of CTs from a specific province. In this case, it is the CMA of Toronto. Current value "CT535"
- **lfssamplesize** : nummber of observations to draw from the LFS for the synthetic populations.
- **niter** : maximum number of external or temperature iterations allowed
- **inerniter** : maximum mber of internal or annealing iterations allowed
- **t0** : this parameter is part of the temperature function (part of the annealing function), used to discern whether a new sample is better than the current sample. Current value is .01.
- **dT** : this parameter is part of the temperature function. The $T0$ variable is reduced by this parameter in each temperature iteration. The Current value is .0002
- **sastop** : is parameter is part of the annealing function. It is used as a criterion for stopping the iterations (substitution of observations in a sample) when an optimal sample has been found. Current value is 0.5
- **tempstop** : is parameter seems unused, i.e., unnecessary. Current value is 0
- **nNoImp** : used parameter. Current value is 100,000
- **nNoImpInner** : sued parameter. Current value is 1,000,000
- **ynstop** : is parameter is used for an optional break point in the algorithm. After processing a number of CTs equal to the parameter, the user is asked whether the algorithm should continue and proceed with more CTs. Current value is 1152

Code chunk 1 (unnamed)

Two main actions

- Initiating the two R libraries used in the script: `readstata13` and `tidyverse`
- Declaring the parameters

The original version of the script used CSV files for the census and the LFS. However, the Stata files offer more metadata and the ability to handle factors (categorical variables) before bringing the data to R. Other

LFS work in ODEN has been done with raw Stata data files, so it was considered better to use the Stata format of the raw LFS files.

The declaration of parameters in this chunk is redundant. The same parameters had been defined in the YAML header. Either way, the parameters can be accessed in code the same way, with the `params$` prefix.

Code chunk 2: setup

Only declaring the default `knitr` options for R Markdown.

Code chunk 3: extract 2016 marginal poercentages for each CT

This chunk performs the data wrangling (cleaning) for the Census 2016 data set. The raw data set is in a long format, where each row contains data specific to a CT and one of 2247 census variables (called dimensions in the data set), and three columns contain the sums of all male, female, and total individuals that fall in those categorical variables for that specific CT and census variable. For instance, a row in the raw data set will have the population between 0-14 years of age of a specific CT, males, females, and both sexes. This long data format is pivoted into a wide data format, where each observation (row) corresponds to a CT, each census variable is has a column, and where only the data for both sexes is included, except for the `Female` variable. The end variables represent percentages of the CT populations that fall within the demographic indicators of the variable.

These are the main steps performed as part of the process:

- Renaming the identifiers for each CT so they follow this pattern: "CT", followed by seven digits, period, followed by two digits (e.g., CT5350063.05)
- Formatting the `Recent_Immigrants` variable
  - Starting with census variable codes 1140 (total immigrant population by CT), 1148 (immigrated between 2006 and 2010), and 1149 (immigrated between 2011 and 2016) and grouping 1148 and 1149 together as "Recent_Immigrants" [5]
- Formatting the `WithPSE` variable (with post-secondary education)
  - Use of census variable 1683 for the number of total number of respondents in a CT who declared their education and variable 1686 for those with "post secondary certificate, diploma, or degree" [6]
- Formatting the `MarriedOrCL` variable (married or common law)
  - Derived from the 59 (population over 15 reporting marital status) and 60 (married or living common law) census variables
- Formatting the age variables:
  - `Total15_24` derived from census variables 14 and 15
  - `Total25_44` from census variables 16, 17, 18, and 19
  - `Total45_64` from variables 20, 21, 22, and 23
  - `Total65_Over` from variable 24
- Formatting the economic family type variables `CouplesWithChildren`, `CouplesWithoutChildren`, `LoneParent`, and `NotInFamilies`, derived from the census variables 82,83,87, and 91, respectively ^[
  - POSSIBLE ERROR: These variables give the percentage of the family types above over the number of couples with children (variable 82). I think variable 81 would have been a better denominator to use (total census families). Also, the variable description in the census file describes these variables as "census families", not economic families. We might have to treat both concepts as the same and hope they are mostly equivalent in practice in order to use the LFS and the census data]
- Formatting the sex variable `Female`, derived from census variable 8 (all ages, males and females) [7]

[5] POSSIBLE ERROR: It looks like the `Recent_Imigrants` variable is supposed to represent the percentage of the population of each CT that is a recent immigrant. However, the variable has many values over 100

[6] POSSIBLE ERROR: The end variable uses variable 1683 as the denominator, but arguably it would be better to use the total population of the CT, which is sometimes different from all the total of respondents who reported their highest education

[7] POSSIBLE ERROR: Census variable 8 does not always sum up to the entire CT population

- Formatting the `EmploymentRate` variable
- Formatting the income variables:
  - `NoEarnings`: From census variable 725
  - `EarnLessThan30`: From variables 728 through 731
  - `EarnMoreThan30`: From variables 732 through 740
  - Using census variable 724 for the denominator (all income responses)
- Formatting the occupation variable `OccupationHealth` from census variable 1890 and the variable 1884 as the occupation total
- Formatting the industry of employment
  - `IndustryRetail`: From census variable 1906
  - `IndustryHealth`: From census variable 1915
  - `IndustryNeither`: From the variable 1897 (industry total) minus the other two specified industries

Code chunk 4 (unnamed)

This code chunk is empty

Code chunk 5: extracting LFS variables

This chunk contains the code that cleans the LFS data set. Unlike the raw census data set, the LFS set is already in a wide format. The unit of observation is the individual respondent and categorical demographic variables are used. For example the variable `AGE_12` has 12 categories, and each respondent falls into one of the 12 categories. Statistical weights are in the variable `FINALWT`, estimating how many individuals of the same characteristics exist in the total population. This is in contrast with the census raw data where the values in the set represent the sum of all individuals in a specific dimension (variable) and CT.

Main steps in the data cleaning

- Filter to have only the province of Ontario (`PROV == 35`)
  - Note: The data wrangling in code chunk 3 did not perform this step and keeps data from all the provinces
- Selecting *ID variables*: record number, CMA, and final weight
- Formatting the same variables as it was done for the census data:
  - The age variables `Total15_24`, `Total25_44`, `Total45_64`, and `Total65_Over` are derived from the categories 1 and 2, 3 through 6, 7 through 10, and 11 and 12 of the LFS `AGE_12` variable
  - The marital status variable `MarriedOrCL` is derived from categories 1 and 2 of the LFS `MARSTAT` variable
  - The education variable `WithPSE` is derived from categories 4 through 6 of the LFS `EDUC` variable
  - The immigration variable `Recent_Immigrants` is derived from category 1 of the LFS `IMMIG` variable
  - The variable `CouplesWithoutChildren` is derived from categories 2,5,8 and 11 from the LFS `EFAMTYPE` variable
  - The variable `CouplesWithChildren` is derived from categories 3,4,6,7,9,10,12 and 13 from the LFS `EFAMTYPE` variable
  - The variable `LoneParent` is derived from categories 14,15,16 and 17 from the LFS `EFAMTYPE` variable
  - The variable `NotInFamilies` is derived from categories 1 and 18 from the LFS `EFAMTYPE` variable
  - The variable `Female` is derived from the category 2 of the LFS `SEX` variable
  - The variable `EmploymentRate` is derived from categories 1 and 2 of the LFS `LFSSTAT` variable
  - The income variables `EarnLessThan30` and `EarnMoreThan30` are obtained from estimating usual annual earnings by multiplying the LFS variables `HRLYEARN` (hourly income rate) and `UHRSMAIN` (usual hours worked in main job)
  - The income variable `NoEarnings` is not used
  - The variable `OccupationHealth` is derived from the category 4 of the LFS `NOC_10` variable

– The variables `IndustryRetail` and `IndustryHealth` are derived from the categories 10 and 17, respectively, of the LFS variable `NAICS_21`

Chunk 6: saved_wrangled

This chunk saves the all the objects in the R environment (mainly the raw and cleaned LFS and census data sets) in the file `clean_data_sets.RData`

Chunk 7: sa func

The code for the `sa.fun` function is included here.

Chunk 8: cma func

This chunk defines the `cma.fun` function, which carries out the following steps and executes the algorithm:

- Cleaning the LFS data set: Use only the Toronto CMA observations and the following variables: REC_NUM; CMA; FINALWT; the variables used for matching LFS and census data (sex, age, married status, education, immigration, family status), employment, industry, income variables.
- Defining the objective function [8]
- Initializing an empty list of CT imputed populations
- Running the `sa.fun` function for each CT (1151 CTs in Toronto) and populate the list of imputed populations
- Applying an optional break in case not all the CTs need to be modeled
- Returning the populated list of CT imputed populations

Chunk 9: sa fit

This chunk sets the seed for the random generation of numbers and executes the algorithm for all of the CTs.

Chunk 10: save SA fit

Saves the product of the algorithm in the file `SA-first_run.rds`

Performance and future improvements

This document includes footnotes of possible errors that should be checked and corrected. The temperature function should also be tested to verify that it works as intended. The values of the parameters can be fine-tuned to improve the accuracy and speed of the algorithm. In one instance, the algorithm processed the 1151 CTs of Toronto in approximately 8 hours. Five of the 1151 CTs could not be modeled. One of these CTs has a population of zero. The reason the other four CTs could not be imputed is unknown.

The algorithm currently draws random samples and swap randomly chosen observations in the LFS sample. The statistical weights in the variable `FINALWT` makes the samples somewhat representative of the population, but also some sort propensity score matching could be used to make the sampling of LFS observations less random and make them match better the characteristics of a specific CTs. Additional variables might be available in the restricted LFS data sets that could be used for this purpose (e.g., CT of the respondent or identification of respondents across time points).

[8] It would be more efficient to do this outside the `cma.fun` function so the function is defined only once

Another room for improvement is the use of consecutive months of the LFS for the sampling and swapping of observations. In this version of the algorithm only the the January 2020 data set is used, but the pool of LFS observations can be widened if the February 2020 and/or the December 2019 data sets were included.

It is hypothesized that the place holder variable for the current sample *curSmpl* is not necessary, and the best sample (*bestSmpl*) and new sample (*newSmpl*) variables alone could be used to run the algorithm. Two decision nodes (logic checks) in the algorithm also seem to be redundant and could be removed. These two changes could help run the algorithm faster. The proposed changes are illustrated in figure 2.

At the level of the R programming language, the algorithm could be run a lot faster if vectorization were used (e.g., the `lapply`) instead of using `for` loops for each CT.

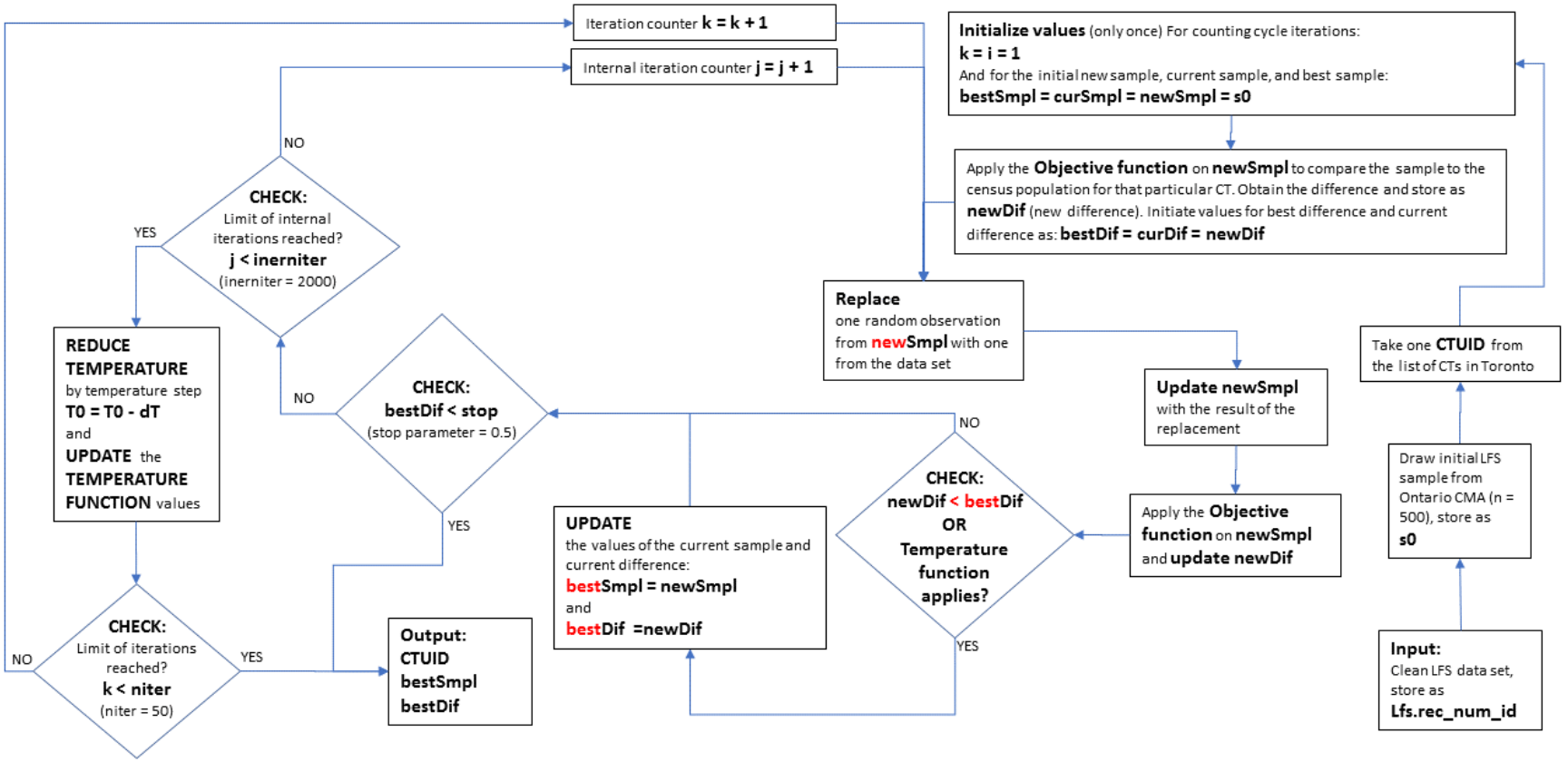

Figure 2: Decision tree diagram for the algorithm with the proposed changes