LivSim Manual

Vikram Kilambi[*1,2], Kevin Bui[†1,2], and Sanjay Mehrotra[‡1,2,3]

[1] *Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL*
[2] *Center for Engineering and Health, Institute for Public Health and Medicine,, Northwestern University Feinberg School of Medicine, Chicago, IL*
[3] *Northwestern University Transplant Outcomes Research Collaborative (NUTORC), Comprehensive Transplant Center, Northwestern University Feinberg School of Medicine, Chicago, IL*

June 2017

[*] Electronic address: vkilambi@u.northwestern.edu; Corresponding author
[†] Electronic address: kbui1993@gmail.com
[‡] Electronic address: mehrotra@northwestern.edu

Contents

1 Background

1.1 Purpose of Software

LivSim is an extensible, open-source discrete-event simulation of the allocation of livers in the US Organ Procurement and Transplantation Network (OPTN). The most recent version, LivSim 1.11, is written for Python 3.4.2 and is designed to work in tandem with the Liver Simulated Allocation Model (LSAM) (v. Aug 2014)[9] as a separate module. LSAM is the standard simulation tool in the transplantation community used to assess alternative liver allocation policies. Unfortunately, hypothetical liver allocation policies that substantially alter the geographic aspects of organ procurement are not implementable in current versions of LSAM. Specifically, LSAM requires that transplant centers be uniquely assigned to a donor service area (DSA) and that DSAs be uniquely assigned to a region or district. Moreover, the source code for LSAM is not publicly available and therefore precludes evaluation of some designs for the OPTN such as that of optimized neighborhoods [6].

The intended purpose of LivSim is to provide an open-source alternative to LSAM that enables the testing of more general geographic structures. Like LSAM, it simulates the liver transplantation waitlist, estimates outcomes of transplant candidates and recipients, and evaluates the performance of liver allocation policies. While LivSim may be used as a standalone simulation environment, it works best when used in conjunction with LSAM input data. This guide aims to familiarize users with the architecture of LivSim.

1.2 Overview of LSAM

For a detailed description of LSAM and its architecture, please refer to the LSAM User's Guide [9]. LSAM is an event-sequenced Monte Carlo simulation of the OPTN. Let t_0 and t_1, where $t_0 < t_1$, be the respective starting time and ending time of an LSAM run. There are two types of input data: parameters describing the structure and policies of the system; and input streams identifying the times for the events such as candidate arrivals, organ arrivals, candidate status progressions, deaths of graft recipients not on the waiting list, relisting of recipients whose grafts fail, and status progressions for relisted graft recipients. Every LSAM run is parameterized by the initial liver transplantation waitlist at t_0 (i.e. a list of candidates on the waitlist in addition to their individual characteristics as of t_0); the geographic relations among transplant centers, DSAs, and regions including the modes of transport, transport distances, and transport times from transplant centers to donor hospitals; allocation rules and sharing policies; organ acceptance models; and post-transplant survival/graft failure models.

LSAM executes a schedule of events in time order through an event handler. These events are generated from schedules collated from three types of input streams. The first is a schedule of arrival times for new candidates joining the waitlist during $(t_0, t_1]$. When a new candidate joins, the candidate's individual characteristics at the scheduled time of listing are also recorded. The second input stream is a schedule of arrival times for new organs during $(t_0, t_1]$. The corresponding donor characteristics at time of donation are recorded. During an organ arrival event, LSAM applies the allocation rules and organ acceptance model to select a candidate on the waitlist to receive an organ. The final input stream is a schedule of status progressions for each candidate on the waitlist during $(t_0, t_1]$. These status changes may include death, waitlist removal for any other reason except transplant, and changes to individual characteristics (e.g. model-for-end stage liver disease [MELD] scores). Moreover, if a candidate receives a transplant, all future status changes for that candidate are nullified. If the candidate is subsequently relisted after receiving a transplant, they will be randomly assigned a status change schedule from a special collection of status change schedules for this purpose.

The Scientific Registry of Transplant Recipients (SRTR) provides default input data based on both historical data and hypothesized models in standard installations of LSAM. Users may also generate their own input data or create new input data using the separate LSAM Candidate Generator and Donor Generator modules. Also, the LSAM user guide provides a detailed description of input generation and highlights some important caveats for users who generate their own input streams, especially from historical data[9].

At the end of a simulation run, LSAM will output the waiting list at t_1, the characteristics of patients who had received grafts as of t_1, the characteristics of candidates who died or were removed from the waiting list for any other reason except transplantation during $(t_0, t_1]$, and the characteristics of donors whose livers were transplanted during $(t_0, t_1]$. LSAM will also produce summary statistics derivable from this information (e.g. number of discards, number of local transplants, pre-transplant mortalities at various MELD thresholds, etc.).

1.3 Other Work

Although LSAM is the simulation environment favored by the clinical community for its comprehensiveness and is the de facto benchmarking tool for liver allocation, it was the operations research community that pioneered the use of discrete-event simulation for modeling OPTN performance. Pritsker et al., as early as 1995, employed an overall architecture that is similar to most implementations thereafter, including LSAM and LivSim [8]. Kreke et al., Shechter et al., and Iyer et al., have all focused on incorporating more accurate biological modeling of individual end-stage liver disease progression into the simulation logic [5, 7, 10]. Taranto et al. and Zenios et al.

developed additional applications for kidney allocation prior to the introduction of the kidney-pancreas simulated allocation model (KPSAM) – the counterpart of LSAM used for kidney allocation [11, 13].

1.4 LivSim versus LSAM

Please refer to the following sections for details regarding LivSim's architecture. LivSim approximates LSAM from information available in publicly released sources. LivSim, unlike LSAM, operates primarily at the DSA/OPO level. The simulation maintains lists of transplant candidates, recipients, and donors; initializes with a starting waitlist; and takes three input streams, which are additions to the liver transplant waitlist, status updates/progressions of waitlist candidates, and arrivals of organs. LivSim then processes each of these events similarly to LSAM.

When candidates arrive to a particular DSA, they are assigned a MELD score, ABO blood type, Status 1 exception (yes or no), and HCC exception (yes or no). During a status progression, LivSim updates the candidate's MELD score and potentially removes the candidate from the waitlist or indicates their death. After a donor arrives, the liver is assigned an ABO blood type and is offered to ABO blood type-compatible candidates in accordance with the sharing policies and geographic structure in place. Moreover, LivSim will determine whether this candidate will relist and, if necessary, return the candidate to the waitlist after calculating the time until graft failure. The candidate is eligible for another transplant after this time, and as in LSAM, relists at a MELD score of 32. Other attributes for candidates and donors may be defined. LSAM (v Aug 2014) input files are compatible with LivSim 1.11 but need to be formatted prior to running LivSim.

When using LSAM input files, LivSim 1.11 will also use LSAM's organ acceptance model to calculate whether a candidate accepts a liver for transplant. It does so by scanning the LSAM input files for the potential recipient's full set of characteristics (instead of only the selected aforementioned characteristics [MELD, ABO, Status1, HCC, etc.]) at the time of the offer and calculates the acceptance probability. If LSAM inputs are unavailable, the user may use a reduced form of LSAM's acceptance model. This reduced model uses LSAM's coefficients for whether the potential recipient is Status 1, the potential recipient's waiting time, whether the potential recipient is listed in the DSA of the procuring OPO, and donor blood type. It also assumes all other patient attributes are held at the baseline. These four characteristics included are also the four most significant predictors in LSAM's acceptance model.

After LivSim finishes running, it will produce the following output:

1. DSA-average MELD at transplant and standard deviation
2. DSA-median MELD at transplant and standard deviation
3. Number of transplants by year and DSA
4. Number of waitlist mortalities by year and by DSA
5. Number of waitlist removals by year and by DSA
6. Average transplant waiting time by year and by DSA
7. Numbers of procured organs directed or received by a specific DSA from each other DSA by year and by DSA

Moreover, if using LSAM input data, LivSim can calculate the following after post-processing:

1. The number of post-transplant mortalities by year
2. Numbers of relist waitlist and re-transplant mortalities by year
3. Average organ transport distances and times for each mode of transport (ground vehicle, helicopter, and airplane) by year
4. Average percentage of organ transported by ground vehicle, helicopter, and airplane by year

Details regarding these calculations are described in later sections. LivSim shares with LSAM several important limitations. Foremost, the existence of biases or omissions in the input streams for patient arrivals, status changes, and donor arrivals affect the quality of the results. For example, users wishing to use historical data must generate hypothetical status progressions for candidates after the actual transplant date as no such status changes would be available in historical data. Additionally, LivSim 1.10 and LSAM do not allow for multiple-organ transplants, listing at multiple centers, and split liver transplants. It is also assumed that the parameter-input data governing the allocation rules and the geographic relationships among transplant centers, donor hospitals, DSAs, and OPOs do not change during a run. Different transplant centers are presumed to

have the same acceptance practices. However, the author believes LivSim is extensible enough to surmount these limitations if necessary.

The primary differences between LivSim and LSAM are the former's ability to incorporate general geographic structures, its focus on OPO/DSA-specific modeling, simplified (but extensible) use of patient and donor characteristics, and reliance on LSAM input data for acceptance modeling and post-processing. Also, LivSim assumes a relist candidate survives until re-transplant.

2 LivSim

2.1 Preliminaries

2.1.1 Installing the Software

LivSim was developed in Python 3.4.2rc1 within the PyCharm Community Edition 4.03 integrated development environment. The user should be able to implement the code in most Python environments. Libraries from standard Python installations include heapq, datetime, operator, pandas sys, queue, csv, and copy. Additional libraries are also needed to run LivSim and are available in most standard scientific installations of Python. These include Numpy 1.9.1 and Scipy 0.14.0.

The source code is organized into the following **.py* files:

- **Standard Files (Require LSAM Acceptance Model)**
  - InputData_LivPlayback_1_11.py
  - allocate.py
  - engine.py
  - entity.py
  - event.py
  - simulate.py
- **Post-Processing Files (Require LSAM Input Files)**
  - Distance_Estimator_Run_2.py
  - OutcomeEstimator_Relists_Regrafts_2.py
  - PostTransplantEstimator_2.py
  - create_new_stats.py
- **Summary Files**
  - compute_vol_diff.py
  - DSA_meld.py
  - mean_diff_summarize.py
  - mean_summarize.py

The most recent versions of the standard files require LSAM's acceptance model. However, this acceptance model can be replaced with a reduced-form model given in the Appendix.

The simulator is called by *simulate.py* and it requires four python files: *allocate.py*, *engine.py*, *entity.py*, and *event.py*. The codes are discussed in great detail in the later subsections. The code *InputData_LivPlayback_1_11.py* loads the input data, and it is called by simulate.py when the simulation initializes.

The files *Distance_Estimator_Run_2.py*, *PostTransplantDeathsEstimator_2.py*, and *OutcomesEstimator_Relists_Regrafts_2.py* are used for average organ transport distances, times, and mode of transport, post-processing to calculate the number of post-transplant and post-re-transplant mortalities by year, and numbers of relists and re-transplants by year. These codes are called altogether by *create_new_stats.py*. Once the new statistics are computed and recorded, the user can call *summarize.py* to create a table comparing relative difference between several cases and a base case. The file *compute_vol_diff.py* calculates the volume gain/loss of several cases relative to a base case for each DSA and records them. These are described in detail in Section 3.

Input File	**Description**
distancetimes.txt	Organ transport distance/time by DSA. Used in post-processing
Donors_Accept.txt	Donor File generated by LSAM or LSAM donor generator without header. Used for acceptance model
Donors.txt	Organ arrival event input data
DSA_AvgTimes.txt	Historical average transport time by DSA
Input_Acceptances.txt	Coefficients for acceptance model
Input_Acceptance_Status1.txt	Coefficients for acceptance model for Status 1 patients
Input_Geography.txt	Geographic structure matrix. Used to define regions, neighborhoods, districts, etc.
Input_Relist.txt	Input distribution for the probability patient will relist
Input_SPartners.txt	Geographical structure matrix. Used to add sharing partners to geographical structure
Patients.txt	Patient arrival event input data
Patients_Accept.txt	Patient file generated by LSAM or LSAM donor generator without header. Used for acceptance model.
Status.txt	Patient status progression event input data.
status_times.txt	Patient status progression event times for post-transplant survival. Used in post-processing
stepsurvival.txt	Step function for post-transplant survival
survivalcoefficients.txt	Coefficients for post-transplant survival model. Used in post-processing
Waitlist_matchmeld.txt	Initial waitlist patient input data

Table 1: LivSim Input Files

2.1.2 Input Data and Formatting

This section describes the input data for LivSim and subsequent post-processing. Table 1 lists the input files used by LivSim. Table 2 lists the DSAs and organ procurement organizations (OPOs) and their corresponding ID numbers in the input and output files of LivSim.

DSA	DSA ID
ALOB-OP1 Alabama Organ Center	0
AROR-OP1 Arkansas Reg. Organ Recovery Agency	1
AZOB-OP1 Donor Network of Arizona	2
CADN-OP1 Donor Network West	3
CAGS-OP1 Sierra Donor Services	4
CAOP-OP1 OneLegacy	5
CASD-IO1 Lifesharing - A Donate Life Org.	6
CORS-OP1 Donor Alliance	7
CTOP-OP1 LifeChoice Donor Services	8
DCTC-OP1 Washington Reg Transplant Community	9
FLFH-IO1 TransLife	10
FLMP-OP1 Life Alliance Organ Recovery Agency	11
FLUF-IO1 LifeQuest Organ Recovery Services	12
FLWC-OP1 LifeLink of Florida	13
GALL-OP1 LifeLink of Georgia	14
HIOP-OP1 Legacy of Life Hawaii	15
IAOP-OP1 Iowa Donor Network	16
ILIP-OP1 Gift of Hope	17
INOP-OP1 Indiana Donor Network	18
KYDA-OP1 KY Organ Donor Affiliates	19
LAOP-OP1 Louisiana Organ Procurement Agency	20
MAOB-OP1 New England Organ Bank	21
MDPC-OP1 The Living Legacy Foundation of MD	22
MIOP-OP1 Gift of Life Michigan	23
MNOP-OP1 LifeSource Upper Midwest OPO	24
MOMA-OP1 Mid-America Transplant Svcs	25
MSOP-OP1 Mississippi Organ Recovery Agency	26
MWOB-OP1 Midwest Transplant Network	27
NCCM-IO1 LifeShare of the Carolinas	28
NCNC-OP1 Carolina Donor Services	29
NEOR-OP1 Nebraska Organ Recovery System	30
NJTO-OP1 NJ Organ and Tissue Sharing Network	31
NMOP-OP1 New Mexico Donor Services	32
NVLV-OP1 Nevada Donor Network	33
NYAP-OP1 Ctr for Donation and Transplant	34
NYFL-IO1 Finger Lakes Donor Recovery Network	35
NYRT-OP1 LiveOnNY	36
NYWN-OP1 Upstate NY Transplant Svcs	37
OHLB-OP1 LifeBanc	38
OHLC-OP1 Life Connection of Ohio	39
OHLP-OP1 Lifeline of Ohio	40
OHOV-OP1 LifeCenter Organ Donor Network	41
OKOP-OP1 LifeShare Transplant Donor Svcs of OK	42
ORUO-IO1 Pacific NW Transplant Bank	43
PADV-OP1 Gift of Life Donor Program	44
PATF-OP1 Center for Organ Recovery and Educ.	45
PRLL-OP1 LifeLink of Puerto Rico	46
SCOP-OP1 LifePoint, Inc.	47
TNDS-OP1 Tennessee Donor Svcs	48
TNMS-OP1 Mid-South Transplant Foundation	49
TXGC-OP1 LifeGift Organ Donation Ctr	50
TXSA-OP1 Texas Organ Sharing Alliance	51
TXSB-OP1 Southwest Transplant Alliance	52
UTOP-OP1 Intermountain Donor Services	53
VATB-OP1 LifeNet Health	54
WALC-OP1 LifeCenter Northwest	55
WIDN-OP1 Wisconsin Donor Network	56
WIUW-IO1 UW Health Organ and Tissue Donation	57

Table 2: DSA Labels and ID Codes

The following provides detail on how each input file is formatted. Single-row tables describe what the various data columns represent. All files are tab delimited unless specified otherwise.

1. distancetimes.txt

DSA ID of Donor Hospital	**DSA ID of Transplant Center**	**Historical Average Transport Distance (miles)**	**Historical Average Transport Time (hours)**	**Transport Mode (*Driving = 0*, *Helicopter = 1*, *Airplane = 2*)**

Table 3: Format of distancetimes.txt Data

There are multiple records for each pair of DSAs corresponding to an alternate donor hospital to transplant center combinations. The source of this data is from LSAM.

2. Donors_Accept.txt

This file is generated by LSAM. Please refer to the LSAM user guide [9] for a description of the columns. This file should have no headers. It is " | " delimited.

3. Donors.txt

Replication #	**DSA ID**	**DSA ID**	**Donor Arrival Time (years)**	**Donor ABO Blood Type** ***(A = 0, AB = 1, B =2, O =3)***	**Organ ID**

Table 4: Format of Donors.txt Data

This input files provides the input stream for organ arrival events by describing the organ arrival time, DSA ID, and donor blood type. The replication number indicates which replication the event will be read by LSAM; the DSA ID indicates which DSA/OPO procures the organ, and arrival time indicates the time LivSim should schedule the event.

IMPORTANT NOTE: It is important that the organ ID for the donor matches the corresponding Donor ID in the LSAM-generated Donors_Accept.txt. LivSim will rely on this correspondence to calculate a high-dimensional acceptance model using LSAM's inputs. Moreover, this file should be sorted by organ arrival times. LivSim also reads each line individually. There is no logic for skipping lines. For example, if the file contains 2 replications of 2-year data, but LivSim is only told to perform 2 replications of 1 year using this file, then LivSim will fail because it will expect the first line after the first replication of the first year to be the first line of the second replication of the first year.

4. DSA_AvgTimes.txt

58×58 matrix of historical average transport times. Entry a_{ij} corresponds to the transport hours from DSA with DSA ID i to the DSA with DSA ID j. The matrix is based on historical OPTN data for 1988-2014 and is not necessarily symmetric.

5. Input_Acceptance.txt

Coefficient	Data	LSAM Variable
-3.9696	Constant	Constant
-0.0021956	Patient	OfferNum
0.00093769	Patient	can_bili
0.13715	Patient	CANHX_EXC_DIAG_HCC2
0.0019133	Patient	can_min_wgt
-0.34554	Patient	CAN_PREV_TX
-0.0033197	Organ	don_bun
-0.00021398	Organ	don_sgpt
0.7619	Patient	local
-0.025285	Patient	traveltime
-0.00023373	Patient	t_CAN_LISTING_DT
0.6677	Organ	don_abo_b
0.75528	Organ	don_abo_ab
-0.11586	Organ	don_protein_urine_yes
0.18977	Organ	don_ebna_pos
0.27843	Organ	don_anti_hyperten_yes
0.10697	Organ	don_insulin_yes
-0.18242	Organ	don_meet_cdc_high_risk_y
-0.54685	Organ	don_non_hr_beat_y
-0.51637	Organ	don_li_biopsy_yes
0.01748	Organ	don_race_p
-0.0074625	Organ	don_race_h
0.048221	Organ	don_death_mech_gunshot
-0.082015	Organ	don_death_circum_natural
0.11223	Organ	don_hist_cancer_no
-0.050175	Patient	can_abo_o
-0.03899	Patient	can_acpt_abo_incomp_y
-0.13962	Patient	can_acpt_li_seg_y
-0.18497	Patient	can_acpt_hbc_pos_y
-0.046836	Patient	can_acpt_hcv_pos_y
0.10345	Patient	can_malig_y
-0.31748	Patient	CANHX_DIAL_PRIOR_WEEK_y
0.043042	Patient	don_pat_gender_match
0.84707	Patient	abo_compat
0.003028	Patient	labmeld
1.0889	Patient	status1b
-0.15437	Patient	don_can_wgt_ratio_gt_p50
0.0024235	Patient	labmeld_gt_p40
0.020667	Patient	labmeld_gt_p50
0.017413	Patient	labmeld_gt_p60
0.050458	Patient	match_meld_gt_p10
-0.040086	Patient	match_meld_gt_p90
-0.015695	Patient	diffmatchlabmeld_gt_p30
-0.0020132	Organ	don_age_in_months_gt_p30
-0.0000687	Organ	don_age_in_months_gt_p70
0.062528	Organ	don_li_biopsy_macro_fat_miss
-0.0000273	Patient	can_max_mile_gt_p40
0.0015365	Patient	can_min_age_gt_p40
0.0017957	Patient	can_hgt_cm_gt_p10
0.00000322	Patient	can_bili_gt_p10
-0.0101	Patient	can_sodium_gt_p10

Table 5: Coefficients of the LSAM's Acceptance Model

This file contains only the numerical values in the leftmost column in Table 5.

6. Input_Acceptance_Status1.txt

Coefficient	Data	Variable
-1.3617	Constant	Constant
0.0018831	Patient	can_min_age
0.0013077	Patient	can_min_wgt
-0.49647	Organ	DON_EXPAND_DON_KI
-0.088581	Patient	traveltime
0.19018	Organ	don_anti_htlv_neg
-0.024067	Organ	don_li_biopsy_yes
-0.012836	Patient	can_acpt_abo_incomp_y
0.015191	Patient	abo_compat
-0.015954	Patient	don_can_hgt_ratio_gt_p40
-0.1603	Patient	don_can_hgt_ratio_gt_p50
-0.049985	Patient	don_can_wgt_ratio_gt_p20
-0.0066923	Organ	don_wgt_kg_gt_p30
0.00011147	Patient	can_bili_gt_p50
0.0048161	Patient	can_bili_gt_p60

Table 6: Coefficients of LSAM's Acceptance Model for Status 1 Patients

This file contains only the numerical values in the leftmost column in Table 6.

7. Input_Geography.txt

This is a 58×58 matrix providing the geographical relationships amongst OPOs that defines regions, districts, or neighborhoods. Entry a_{ij} takes value 1 if DSA with DSA ID j shares with the DSA with DSA ID i when an organ is procured in the latter during regional allocation. This matrix is not available in LSAM in full generality and hence was the reason LivSim was created.

8. Input_Relist.txt

This file contains the lower bound, upper bound, and mean for the uniformly distributed probability that a transplanted patient will be relisted. The current values are based on averages computed from historical OPTN data 1988-2014.

9. Input_SPartners.txt

This is a 58×58 matrix adding DSA sharing partners to existing geographical relationships amongst OPOs. Entry a_{ij} takes value 1 if DSA with DSA ID j shares with the DSA with DSA ID i when an organ is procured as a sharing partner. Although this matrix is read by LivSim, it is not required to actually use it unless the user wishes to study sharing partners.

10. Patients.txt

Replication #	Patient ID	DSA ID	DSA ID	Patient Arrival Time (years)	Patient ABO Blood Type *(A=0, AB=1, B =2, O =3)*	Patient Allo-cation MELD	Patient Lab MELD	Patient HCC Status *(No = 0, Yes = 1)*	Status1 *(No=0, Yes =1)*	Sodium Score	Inactive *(No = 0, Yes = 1)*

Table 7: Format of Patients.txt Data

This input files provides the input stream for patient arrival events by describing the patient arrival time, DSA ID, patient blood type, starting allocation MELD (6-40), starting lab MELD, whether patient receives an HCC exception, whether the patient receives a Status 1 exception, starting lab sodium value, and whether patient is inactive. Status 1A and Status 1B patients are treated identically. The replication number indicates which replication the event will be read by LSAM; the DSA ID indicates which DSA/OPO lists the patients, and arrival time indicates the time LivSim should schedule the event.

IMPORTANT NOTE: It is important that the patient ID for the patient matches the corresponding candidate ID in the LSAM-generated Patients_Accept.txt. LivSim will rely on this correspondence to calculate a high-dimensional acceptance model using LSAM's inputs. Moreover, this file should be sorted by patient arrival times. LivSim also reads each line individually. There is no logic for skipping lines. For example, if the file contains 2 replications of 2-year data, but LivSim is only told to perform 2 replications of 1 year using this file, then LivSim will fail because it will expect the first line after the first replication of the first year to be the first line of the second replication of the first year.

11. Patients_Accept.txt

This file is generated by LSAM. Please refer to the LSAM user guide [9] for a description of the columns. This file should have no headers. It is " | " delimited.

12. Status.txt

Replication #	Patient ID	Status Event Time (years)	Dies *(No=0, Yes =1)*	Removed from Waitlist *(No = 0, Yes = 1)*	Updated Alloca-tion MELD	Updated Lab MELD	Updated Sodium MELD	DSA ID	DSA ID	Updated Inactive Status *(No = 0, Yes = 1)*

Table 8: Format of Status.txt Data

This input files provides the input stream for status progression events for a particular patient. Each row updates a given patient's MELD scores, sodium scores, and inactive statuses at a particular DSA. Additionally, a status update event may indicate that the particular patient dies or is removed from the waitlist. Death or waitlist removal nullifies all other updates to MELD scores, sodium, etc., at the time of the death or removal and afterwards. The replication number indicates which replication the event will be read by LSAM; the DSA ID and patient ID indicates which patient is to be updated, and the status event time indicates the time LivSim should schedule the update event.

IMPORTANT NOTE: This file should be sorted by status event times. LivSim also reads each line individually. There is no logic for skipping lines. For example, if the file contains 2 replications of 2-year data, but LivSim is only told to perform 2 replications of 1 year using this file, then LivSim will fail because it will expect the first line after the first replication of the first year to be the first line of the second replication of the first year.

13. status_times.txt

This file is used only for post-processing. The first column is the patient ID and the second column is the status event time (years). *These columns should match the second and third columns from Status.txt.*

14. stepsurvival.txt

Step Probability	**Days Survived After Transplant**	**Group Probability**

Table 9: Format of stepsurvival.txt Data

This file is used only for post-processing for post-transplant outcomes. Given individual characteristics, a step probability is calculated that determines the number of days survived after transplant. Group probability can be ignored. The source of this data is from LSAM's post-transplant survival model.

15. survivalcoefficients.txt

Coefficient	Data	LSAM Variable
0.096521	Organ	don_race_black
0.231046	Organ	don_race_hispanic
0.136725	Organ	don_race_not_wbh
0.115276	Organ	don_cod_cerebro_stroke
0.495574	Organ	don_non_hr_beat_y
0.071974	Organ	don_li_biopsy_y
0.135386	Organ	don_diab_y
0.104547	Organ	don_anti_hcv_pos
-0.18468	Organ	don_age_lt_18
0.069699	Organ	don_age_40_50
0.215867	Organ	don_age_50_60
0.442272	Organ	don_age_60_70
0.53747	Organ	don_age_ge_70
0.048991	Organ	don_hgt_per_10cm_dec
0.187979	Patient	can_race_black
-0.12496	Patient	can_race_hispanic
-0.11149	Patient	can_race_not_wbh
0.357815	Patient	can_life_support_y
0.117489	Patient	can_prev_abdom_surg_y
0.178723	Patient	can_prev_abdom_surg_m
0.296156	Patient	can_dgn_hcv
-0.11892	Patient	can_dgn_chol
0.019462	Patient	can_dgn_ahn
0.03673	Patient	can_dgn_met_dis
0.445674	Patient	can_dgn_mal_neo
0.118069	Patient	can_dgn_other
0.206299	Patient	can_dial_y
0.111769	Patient	can_prev_malig_m
0.157912	Patient	can_prev_malig_y
0.476963	Patient	can_prev_li
-0.22144	Patient	can_ln_albumin
0.040675	Patient	can_albumin_m
0.140549	Patient	can_ln_creat
0.349983	Patient	can_creat_m
-0.10011	Patient	can_age_lt_18
0.21622	Patient	can_age_18_25
0.002244	Patient	can_age_25_35
-0.10293	Patient	can_age_45_55
-0.02486	Patient	can_age_55_65
0.165089	Patient	can_age_ge_65
0.069976	Patient	regional
0.184682	Patient	national
0.185855	Patient	can_diab_ty_any
0.225524	Patient	can_growth_fail
0.157561	Patient	can_ascites_y
0.155651	Patient	can_portal_vein_y

Table 10: Coefficients of LSAM's Post-Transplant Survival Model

This file is used only for post-processing for post-transplant outcomes. The leftmost column of numerical coefficients is from LSAM's post-transplant survival model.

16. Waitlist_matchmeld.txt

Patient ID	**DSA ID**	**Patient Arrival Time (years)**	**ABO Blood Type** *(A = 0, AB = 1, B =2, O =3)*	**Patient Starting MELD**	**Patient HCC Status** *(No = 0, Yes = 1)*	**Status1** *(No = 0, Yes = 1)*	**Sodium Score**	**DSA ID**	**Inactive** *(No = 0, Yes = 1)*

Table 11: Format of Waitlist_matchmeld.txt Data

This input files provides the initial waitlist by describing the patient arrival time, DSA ID, patient blood type, starting MELD, whether patient receives an HCC exception, whether the patient receives a Status 1 exception, starting lab sodium value, and whether patient is inactive. Status 1A and Status 1B patients are treated identically. Waiting times are negative numbers, indicating that patient arrived before initialization of the simulation at time 0. The characteristics describe the patient at the initialization of the simulation. Starting MELD values may be either lab MELD or allocation MELD values depending on the user's preferences.

2.1.3 InputData_LivPlayback_1_11.py

This file loads and shapes the input data for use by the simulation. It is called by simulate.py during the initialization of the simulation. In the setting section in the code, there are variables that can be adjusted to change the setting of the simulation. The variable *i_initial* toggles whether an initial waitlist should be loaded (all other input data will be loaded). The variable *exclude_hi_pr* allows an option to exclude Hawaii and Puerto Rico from the simulation. After the setting section, the input files mentioned in the preceding subsection are read. As described in the following section, LivSim treats patients and donors as class objects. The OPTN is represented as a list of 58 lists where each list corresponds to the transplant-candidate waiting list at a particular DSA. Loading the initial waitlist instantiates patient objects with characteristics as described by the columns of Table 11. These objects are then added to a list in the OPTN data structure.

When reading input data, LivSim follows certain conventions regarding missing values, values out of range, etc.:

1. Individuals with empty sodium scores are assigned sodium values of 137 (the highest effective sodium value as per UNOS guidelines at time of writing).
2. Allocation MELD scores range from 6-40.
3. Status 1 candidates are given allocation MELD scores of 41. This is a programming convention.
4. Lab-, allocation-, and sodium-MELD scores are assigned based on the options selected (discussed in the next subsection).
5. If patients receive HCC exceptions, they are assigned MELD scores based on the HCC MELD schedule and selected options (discussed in the next subsection).

2.1.4 Running the Software

This subsection describes running LivSim. Post-processing is described in Section 3. After ensuring that the input files have been formatted properly, the user needs to execute the following steps:

0. For Windows user, go to `https://www.continuum.io/downloads#windows` to download Anaconda.
1. Mac or Linux:
   - Open command terminal and change to the directory containing the Standard Files (ref. pg. 5)

   Windows:
   - Open Anaconda Prompt and change to the directory containing the Standard Files (ref. pg. 5).
2. Type and enter the following :

   ```
   python simulate.py {maxtime} {nreps} [{regional_sharing}, {sodium}, {capanddelay}, {spartners}] {ShareU} {ShareL} {BOOST} {D}
   ```

   where
   - maxtime: Desired run-length (years)
   - nreps: Number of desired replications
   - regional_sharing: Toggles full-regional allocation (i.e. no local allocation) if value is 1.
   - sodium: Applies MELD-Na in lieu of traditional MELD if value is 1.
   - capanddelay: Applies cap-and-delay policy for HCC exceptions value is 1.
   - spartners: Toggles sharing-partner allocation if value is 1.
   - ShareU: Upper MELD threshold in liver allocation.
   - ShareL: Lower MELD threshold in liver allocation.
   - BOOST: Number of boost points awarded to allocation MELD awarded to local candidates during allocation.
   - D: Directory where the output files will be saved in.

   Example:

   python simulate.py 5 5 [0,1,1,1] 35 15 5 “C:/Users/JohnDoe/Documents/ ”

   This line runs the simulator for 5 years in 5 replications It implements the sodium policy, the cap-and-delay policy, and the sharing partners policy but no regional sharing policy. The MELD Policy is Share35/Share 15 with local boost of 5 points. The output files are saved in C:/Users/JohnDoe/Documents/.

The user will receive messages with time stamps for when the simulation input and initialization are complete, after each replication has finished, and when the simulation terminates.

2.1.5 Output Formatting

This section describes the output files generated by LivSim. The output files are saved as csv files. Output files are usually organized by year and by replication number. The following provides detail on how each output file is formatted. Single-row tables describe what the various data columns represent. All files are comma delimited unless specified otherwise.

1. Output_deaths.txt

# of Deaths	Year	Replication #

Table 12: Format of Output_deaths.csv Output

2. Output_mr_disparity_mean.csv

DSA Average Mortality Rate	Year	Replication #

Table 13: Format of Output_mr_disparity_mean.csv Output

DSA average mortality rate is calculated as the ratio of the number of deaths in a year to the sum of the waitlist arrivals that year and number of candidates at the start of the year.

3. Output_mr_disparity_std.csv

DSA Mortality Rate Standard Deviation	Year	Replication #

Table 14: Format of Output_mr_disparity_std.csv Output

Calculated as the standard deviation of the DSA average mortality rates.

4. Output_meld_disparity_mean.csv

DSA Transplant MELD Average	Year	Replication #

Table 15: Format of Output_meld_disparity_mean.csv Output

Output is the Average MELD at transplant over the year for non-Status 1 candidates averaged across DSAs.

5. Output_meld_disparity_std.csv

Standard Deviation of Average DSA Transplant MELD	Year	Replication #

Table 16: Format of Output_meld_disparity_std.csv Output

Output is the standard deviation of DSA-average MELD at transplant for the year across DSAs (for non-Status 1 candidates).

6. Output_meld_median_mean.csv

DSA Transplant MELD Median	Year	Replication #

Table 17: Format of Output_meld_median_mean.csv Output

Output is the Median MELD at transplant over the year for non-Status 1 candidates averaged across DSAs.

7. Output_meld_median_std.csv

Standard Deviation of DSA Transplant MELD Median	Year	Replication #

Table 18: Format of Output_meld_median_std.csv Output

Output is the standard deviation of DSA-median MELD at transplant for the year across DSAs (for non-Status 1 candidates).

8. RawOutput_ydeaths.csv

Year	Replication #	Replication #	Deaths in DSA ID 0	Deaths in DSA ID 1	...	Deaths in DSA ID 57

Table 19: Format of RawOutput_ydeaths.csv Output

9. RawOutput_ytransplants.csv

Year	Replication #	Replication #	Transplants in DSA ID 0	Transplants in DSA ID 1	...	Transplants in DSA ID 57

Table 20: Format of RawOutput_ytransplants.csv Output

10. RawOutput_yarrivals.csv

Year	Replication #	Replication #	Patient Arrivals in DSA ID 0	Patient Arrivals in DSA ID 1	...	Patient Arrivals in DSA ID 57

Table 21: Format of RawOutput_yarrivals.csv Output

11. RawOutput_ycandidates.csv

Year	Replication #	Replication #	Candidates in DSA ID 0	Candidates in DSA ID 1	...	Candidates in DSA ID 57

Table 22: Format of RawOutput_ycandidates.csv Output

Output is the number of candidates at the beginning of the year.

12. RawOutput_yremoved.csv

Year	Replication #	Replication #	Patients Removed in DSA ID 0	Patients Removed in DSA ID 1	...	Patients Removed in DSA ID 57

Table 23: Format of RawOutput_yremoved.csv Output

Output is the number of waitlist candidates removed from waitlist during the year due to any reason except death or transplant.

13. RawOutput_ywait.csv

Year	Replication #	Replication #	Accumulated Total Transplant Waiting Time in DSA ID 0	Accumulated Total Transplant Waiting Time in DSA ID 1	...	Accumulated Total Transplant Waiting Time in DSA ID 57

Table 24: Format of RawOutput_ywait.csv Output

Individual transplant waiting time is calculated as the difference in simulation clock times (years) between time of patient listing and time of transplant. Accumulated total transplant waiting time is the sum of all such waiting times of patients transplanted in that DSA during the year. The user must divide by the number of transplants to obtain the average waiting time per transplant.

14. RawOutput_yMELD.csv

Year	Replication #	Replication #	Accumulated Total Transplant MELD in DSA ID 0	Accumulated Total Transplant MELD in DSA ID 1	...	Accumulated Total Transplant MELD in DSA ID 57

Table 25: Format of RawOutput_yMELD.csv Output

Accumulated total transplant MELD is the sum of all MELD scores of patients transplanted in that DSA during the year. The user must divide by the number of transplants to obtain the average MELD per transplant.

15. RawOutput_DSAs.csv

This is a 58×58 matrix summarizing organ sharing across all years and replications. Entry a_{ij} represents the number of livers procured from DSA with DSA ID i that were transplanted in the DSA with DSA ID j across all years and replications.

16. RawOutput_DSAs2.csv

This is a $58 \times 58 \times N$ array where N is the number of replication-years (i.e. a 5-year 5-replication run yields 25 replication-years) summarizing organ sharing. Entry $a_{ij}(t)$ represents the number of livers procured from DSA with DSA ID i that were transplanted in the DSA with DSA ID j up to replication-year t. This array is provided in addition with the RawOutput_DSAs.csv so that the user may analyze organ sharing patterns over time if they so wish to.

17. RawOutput_removals.csv

Year	Replication #	Removal Time	Removed Patient ID	Patient Allocation MELD	Patient Lab MELD

Table 26: Format of RawOutput_removals.csv Output

Output file contains information of any patient removed for any reason other than transplant/death for further analysis.

18. RawOutput_TxID.csv

Year	Replication #	Transplant Time	Transplant Patient ID	Regional Transplant *(No = 0, Yes = 1)*	National Transplant *(No = 0, Yes = 1)*

Table 27: Format of RawOutput_TxID.csv Output

Output file contains information of any patient transplanted for further analysis. This does not include those who were ever or would have been relisted. Regional transplant refers to non-local but regional transplant (i.e. organ came from same region, district, or neighborhood).

19. RawOutput_DoID.csv

Year	Replication #	Transplant Time	Transplant Patient ID	Donor ID

Table 28: Format of RawOutput_DoID.csv Output

Output file contains information of any patient transplanted (and link to corresponding donor) for further analysis. This does not include those who were ever or would have been relisted.

20. RawOutput_yrelists.csv

Year	Replication #	Replication #	Relists in DSA ID 0	Relists in DSA ID 1	...	Relists in DSA ID 57

Table 29: Format of RawOutput_yrelists.csv Output

Output is the number of candidates that relisted for transplant during the year by DSA.

21. RawOutput_yregrafts.csv

Year	Replication #	Replication #	Regrafts in DSA ID 0	Regrafts in DSA ID 1	...	Regrafts in DSA ID 57

Table 30: Format of RawOutput_yrelists.csv Output

Output is the number of relisted candidates that received a re-transplant during the year by DSA.

22. RawOutput_TxIDregraft.csv

Year	Replication #	Re-Transplant Time	Re-Transplant Patient ID	Regional Re-Transplant *(No = 0, Yes = 1)*	National Re-Transplant *(No=0, Yes = 1)*

Table 31: Format of RawOutput_TxIDregraft.csv Output

Output file contains information of any patient re-transplanted for further analysis. Regional transplant refers to non-local but regional transplant (i.e. organ came from same region, district, or neighborhood).

23. RawOutput_DoIDregraft.csv

Year	Replication #	Re-Transplant Time	Re-Transplant Patient ID	Donor ID

Table 32: Format of RawOutput_DoIDregraft.csv Output

Output file contains information of any patient re-transplanted (and link to corresponding donor) for further analysis.

24. RawOutput_Relistid.csv

Year	Replication #	First Trans-plant Time	Patient ID	Patient Allocation MELD at First Transplant Time	Patient Earliest Re-Transplant Time

Table 33: Format of RawOutput_Relistid.csv Output

Output file contains information of any patient re-listed. As described below, whether a patient will relist is determined at time of the first transplant. The earliest re-transplant time is the time that the first graft will fail and that the patient will be eligible for a re-transplant.

2.2 Architecture

2.2.1 Simulation Engine

Figure 1 depicts the schematic for LivSim. The engine first initializes with all global variables and option setting. Next, it calls InputData_LivPlayback_1_11.py to read the input data and sets the pointers for reading the patient arrival, organ arrival, and status progression input streams.

Afterward, the replication is initialized (clock is set to starting value and replication statistics are initialized). Prior to processing the input streams for any replication, the starting waitlist is copied from memory. The engine then reads the lines corresponding to the pointers for the patient arrival, organ arrival, and status progression input streams. Using the event times in the input files, it then determines the next event (or whether the next event is the end-of-year event). Based on the determination, the engine generates an event notice for the event and passes relevant data for the event (e.g. event type, event time, DSA event occurs, etc.). The event notice is added to a queue (called the calendar) and the pointer for the input stream from which the event was read is advanced one line. Event notices are added to the calendar until all events for the replication have been scheduled.

Lastly, once all events have been loaded onto the calendar, the engine continuously pops an event from the calendar and calls a function for the particular event as described by the event notice until it is empty. After the calendar becomes empty, the end-of-replication event is triggered and replication statistics are cleared. If all replications have finished, the engine writes all of the output files and the simulation terminates.

2.2.2 OPTN Data Structure

The OPTN data structure is a list of lists with dimension n, where n is the number of DSAs. Each element contains a list of patient-class objects that represent the candidates listed at a particular DSA. At the beginning of each replication, the data structure is initialized with a copy of the starting waitlist. During organ allocation, this data structure is temporarily copied (to pass by value in Python).

2.2.3 Engine Classes (engine.py)

This section discusses the constructed classes in engine.py.

1. **Event Class**
   This class of objects is that of the event notices that are loaded onto the calendar. They have three methods: (1) event type (e.g. Organ Arrival), (2) event time (time to be executed on the simulation clock), and (3) event information (any data that needs to passed to the event function). By default, the engine will pass the entire line read from the input stream as the event information. The user can modify the event information method to pass additional information, characteristics, etc., when the calendar calls the function associated with the event type. Provided that each line of the input file for the corresponding stream has the necessary extra column data, users can refer to the appropriate column of the event information to pass additional information.

2. **G Class**
   LivSim follows the convention of maintaining all global variables in a single object class G. Class G contains all the options listed in Section 2.1.4 in addition to the array dimensions of the output files listed in Section 2.1.5.

3. **SimStat Class**
   This class is used to maintain performance measures and statistics during a replication. Its individual methods are the statistics such as the number of transplants that occurred during that replication-year and number of patient arrivals during that replication-year. Instantiation of this class may be modified by other events and during the end-of-year event, but each such instantiation is deleted at the end of the replication.

2.2.4 Entity Classes (entity.py)

This section discusses the entity classes constructed for patients and organs in entity.py

1. **Patient Class**
   Every instantiation of this class represents a patient. Upon construction, they are endowed with attributes for the patient's *id*, *DSA*, and *create_time*. Patient objects are created during the *Patient Arrival Event* and creation of the initial waitlist. Information to populate the methods and attributes comes from the *event information* when scheduling these events and thereby the input files corresponding to the patient arrivals and initial waitlist. The attributes *id* and *DSA* give the patients their unique identifiers and

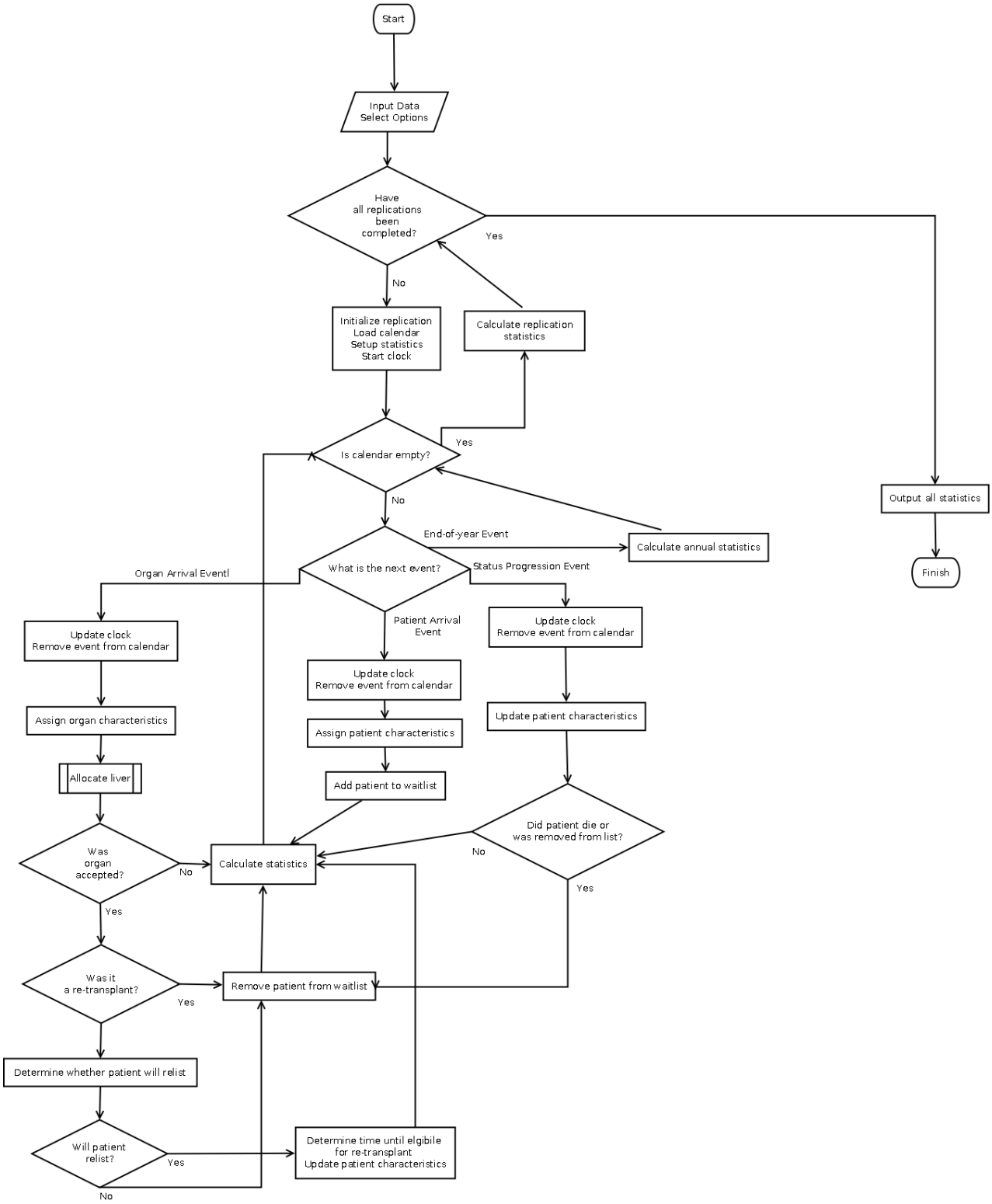

Figure 1: LivSim Architecture

locations and are used to refer to the patients during various function calls. The attribute *create_time* gives in simulation clock units, the time the patient object was created (i.e. listed). Other attributes describe patient characteristics and they include ABO blood type; allocation MELD score; lab MELD score; HCC exception status; Status1; sodium score; waitlist inactive status; whether the patient is a relisted patient; and the time at which a patient is eligible for a re-transplant (*relistTxTime*).

2. **Organ Class**
   Every instantiation of this class represents an organ. Upon construction, they are endowed with attributes for the organ's *id* and *DSA*. Organ objects are created during the *Organ Arrival Event* and exist only until the organ is either allocated or discarded. Information to populate the methods and attributes comes from the *event information* when scheduling these events and thereby the input files corresponding to the organ arrivals input stream. The attributes *id* and *DSA* give the organs their unique identifiers and locations and are used to refer to the donors during various function calls. Additionally, organs are endowed for an attribute for ABO blood type.

2.3 Events and Subordinate Functions

This section summarizes the various events processed by LivSim. These events are modeled as functions in event.py. As described above, LivSim executes each event as a function call when the corresponding event notice is retrieved from the calendar. Each event/function takes the *event information* as an input (and does not necessarily return anything). Furthermore, this section discusses how the organ is allocated within the simulation through the functions in allocate.py.

2.3.1 Arrival(Patient)

Patient arrival is modeled through the *Arrival* function in event.py. This event instructs LivSim to create a patient with particular characteristics and add the patient to the waitlist. First, a patient object is created and assigned an *id*, *DSA*, and *create_time* in addition to Status 1 (yes/no), ABO blood type, HCC exception status, sodium value, lab MELD, allocation MELD, and waitlist active inactive status as per the function input.

LivSim will use allocation MELD to prioritize patients for transplantation. The user may wish to incorporate sodium MELD (MELD-Na) in one of two ways (1) ensure allocation MELD corresponds to MELD-Na in input streams or (2) toggle the *sodium* option in the global variables. If using the latter, the simulation will first truncate sodium values to the interval $[125, 137]$ and use lab MELD to calculate MELD-Na according to the following equation [2]:

$$\text{MELD Na} = \text{LabMELD} + 1.32 \times (137 - \text{Na}) - (0.033 \times \text{LabMELD} \times (137 - \text{Na})). \tag{1}$$

Additionally, all allocation MELD scores (regardless of the sodium update) are rounded to the nearest integer and truncated to the interval $[6, 40]$. By convention, Status 1 candidates are assigned an allocation MELD of 41. If the patient is receiving an HCC exception, then either the patient retains their allocation MELD score. If the *capanddelay* option is also toggled, then patient's allocation MELD is the minimum of their lab MELD or 28[3]. After allocation MELD is assigned, the patient is then added to the OPTN data structure (at the place corresponding the patient's DSA) and statistics regarding the number of arrivals for the year and number of candidates currently in the OPTN are updated.

2.3.2 Progression

Progression is modeled through the *Progression* function in event.py. This event instructs LivSim to retrieve a particular patient from the OPTN data structure and update the patient's characteristics. These updates are passed in the *event information* for this event and thereby from the progression/status change input stream data. First, LivSim searches the element of the OPTN data structure corresponding to the patient's DSA for the patient object. If the patient has relisted for transplant, then this event is skipped. Additionally, if the patient is indicated to die or be removed from the waitlist, the patient object is deleted; statistics regarding the number of deaths/removals that year and the numbers of candidates currently in the OPTN are all updated; and the event then terminates.

Otherwise, the patient's lab MELD, allocation MELD, sodium, and inactive statuses are updated in accordance with the *event information*. If the *sodium* option is toggled, sodium values will be truncated to the interval $[125, 137]$ and allocation MELD will be then updated according to Equation (1) for non-HCC and non-Status 1 candidates (and subsequently rounded to the nearest integer in $[6, 40]$. If the *capanddelay* option is toggled, HCC patients' allocation MELD will be updated according to the schedule in Table 34 [2].

Waiting Time	Allocation MELD Update
$0 - 0.5$ years	Maximum of 28 or old allocation MELD
$> 0.5 - 0.75$ years	Maximum of 29 or old allocation MELD
$0.75 - 1.00$ years	Maximum of 31 or old allocation MELD
$> 1.00 - 1.25$ years	Maximum of 33 or old allocation MELD
$> 1.25 - 1.50$ years	Maximum of 34 or old allocation MELD
> 1.50 years	Maximum of 40 or old allocation MELD+1

Table 34: Allocation MELD Updates for HCC Patients with Cap and Delay

2.3.3 Organ Arrival

Organ arrival is modeled through the *OrganArrival* function in event.py. This event instructs LivSim to create an organ with particular characteristics and attempt to allocate the organ. First, an organ object is created and assigned an *id*, *DSA*, and ABO blood type using the *event information*. This event then calls the *Allocate* function and the dependent *MatchRun*, *MatchCheck*, and *Offer* functions that allocate the organ. Organ allocation in LivSim follows the schematic shown in Figure 2.

If a candidate for the organ is not found, the event ends. Otherwise, the *Allocate* function will return the *id* and *DSA* of the accepting patient for transplant. The corresponding patient object is retrieved from the OPTN data structure. If this is the 1st transplant for the patient, then using the uniform distribution provided in Input_Relist.txt, LivSim determines whether the patient will ever relist. If the patient will not relist or just received a re-transplant, then the patient object is deleted from the OPTN data structure, statistics regarding the number of transplants, transplant MELD, waiting time, local/national transplant, supplying DSA, and transplanting DSA are recorded, and the event ends (separate statistics are kept for re-transplants). If the patient relists, the patient object's *Relist* attribute is updated to the value 1 and the *RelistTxTime* attribute is assigned a value equal to the simulation clock plus an increment. The increment represents the time until the first graft fails and whereby the patient becomes eligible for a re-transplant. This increment is assigned with the empirical distribution based on OPTN data in Table 35 [1].Thereafter, the patient is assigned an allocation MELD of 32 (as is the convention in LSAM) and the event concludes.

Increment (Graft Failure Time)	Probability
5 years	0.60
2 years	0.20
1 year	0.20

Table 35: LivSim First Graft Failure Times for Re-Transplant

2.3.4 The Allocate, MatchRun, MatchCheck, and Offer Functions

This section discusses all of the functions in allocate.py.

The *Allocate* function is called by the *OrganArrival* event. The function takes the organ object passed from the *OrganArrival* event and returns a dummy value indicating whether the organ was unable to be allocated or a list containing the the patient id and DSA of an accepting patient. The purpose of the function is to create the match-run list for the allocation (the offer list), pass it to the *MatchRun* function, and return the results from *MatchRun*. First, this function copies the element of OPTN data structure corresponding to the DSA where the organ was procured (the local list). It adds any local score boosts (if specified in the options). Second, it copies the elements of the OPTN data structure corresponding to the DSAs in the region/district/neighborhood/sharing partner community for regional allocation and applies any regional score boosts if applicable (the regional list). Other remaining elements are copied (the national list).

The next steps depend on the sharing policy specified. If *regional_sharing* is toggled, the local and regional lists are combined, sorted by allocation MELD, and then added to a sorted national list to yield the offer list. By default, LivSim implements the Share 15 and Share 35 policies for liver allocation [4, 12]. However, the user can specified the thresholds by changing the values for *ShareU* and *ShareL*. The local and regional lists will be partitioned into three lists: one having candidates with allocation MELD $\geq$ *ShareU*, one with allocation MELD in [*ShareL*,(*ShareU*-1)], and one with allocation MELD $<$*ShareL*. The national list will be partitioned into a list of candidates with allocation MELD $\geq$ *ShareL* and a list with candidates with allocation MELD $<$*ShareL*. These lists are then sorted and recombined to yield the offer list that matches the policy.

The second function, *MatchRun*, takes the offer list and organ object from the *Allocate* function, executes the offers, and returns the results back to the *Allocate* function. A local variable, *noffers*, counts how many offers

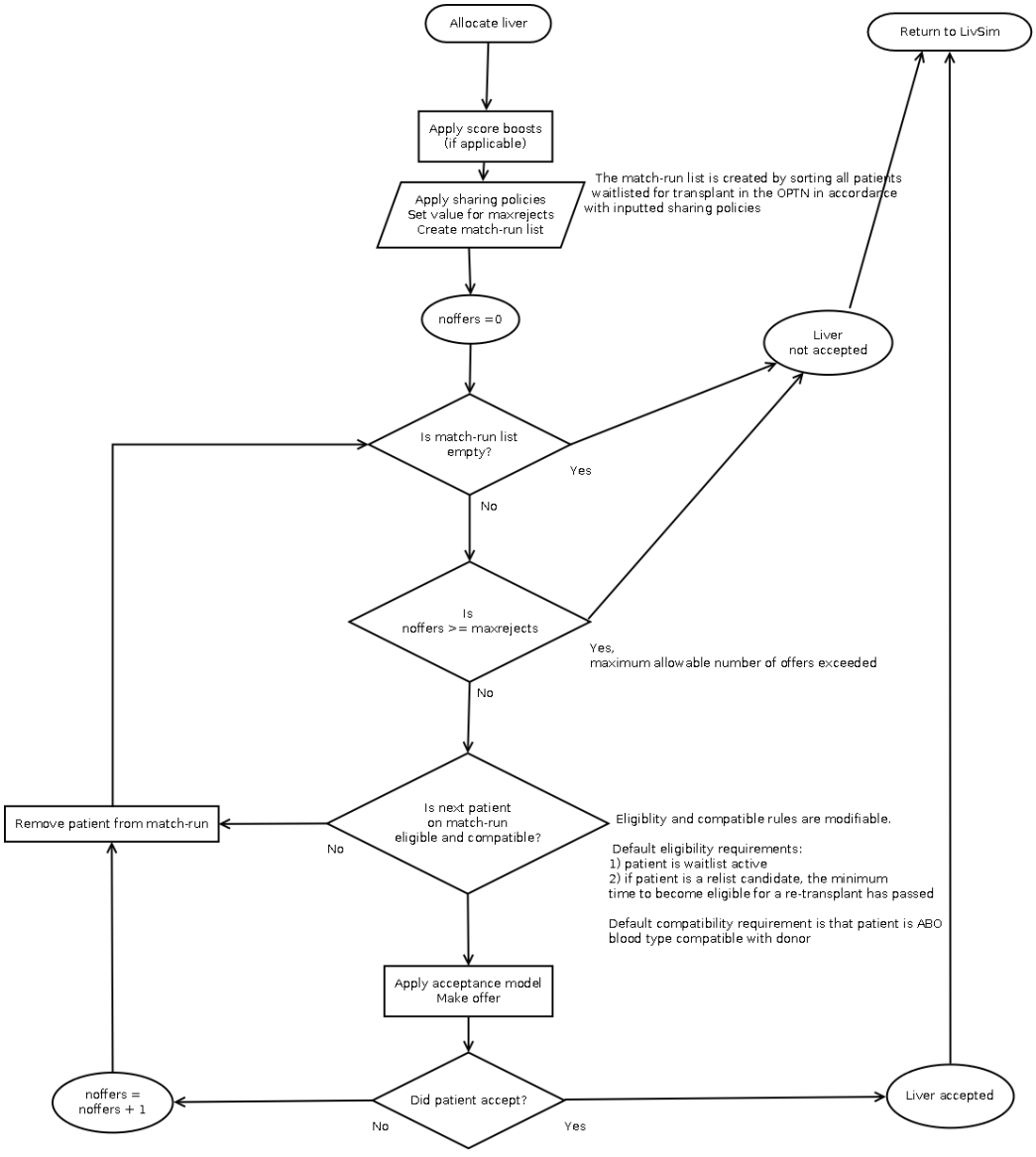

Figure 2: Organ Allocation in LivSim

have been made. The global variable, *maxrejects*, specifies the maximum number of offers that can be made before the organ is discarded. While offers can be made, the *MatchRun* traverses the patient objects on the offer list. When doing this, it first calls the third function, *MatchCheck*, and passes the current patient object. The *MatchCheck* function's purpose is for compatibility/cross-matching/eligibility checking of patient and organ. It returns the value 0 if the patient is incompatible or ineligible or 1 if the patient is eligible in compatible. The user may specify whatever criteria they wish, but by default, the *MatchCheck* function checks whether the patient is ABO blood type compatible with the organ, the patient does not have an inactive waitlist status, and the patient has or will relist, that is his or her first graft has failed (i.e. the simulation clock has exceeded *RelistTxTime*).

If *MatchCheck* returns 0, then *MatchRun* will continue traversing the patient objects on the offer list. If *MatchCheck* returns 1, then it will make an offer to the patient by calling the *Offer* function. The *Offer* function's role is to take the offered organ and current patient object as input, apply the acceptance model, and return the results to *MatchRun*.

IMPORTANT: The default acceptance model is reproduction of LSAM's acceptance model and requires LSAM files to run (Donors_Accept.txt, Input_Acceptance.txt, Input_Acceptance_Status1.txt, Patients_Accept.txt). Using the *id* attributes contained in the donor and patient objects, this function will retrieve additional characteristics of the organ and patient from the LSAM file that are not explicitly modeled in LivSim. If the vector of patient and organ characteristics are denoted as x (from Donors_Accept.txt and Patients_Accept.txt) and the coefficients are denoted as β (from Input_Acceptance.txt and Input_Acceptance_Status1.txt), then the acceptance probability is calculated using standard logistic regression

$$\text{Accept Prob} = \frac{\exp(\beta^T x)}{1 + \exp(\beta^T x)}. \tag{2}$$

A separate set of patient and organ characteristics and coefficients are used for Status 1 patients; however the acceptance probability is also calculated following Equation (2). Once the acceptance probability is calculated, a random uniform number is generated. If the random number is less than or equal to Accept Prob, then the organ is accepted. Otherwise, it is rejected by the patient. A reduced-form acceptance model not requiring LSAM's inputs is provided in the Appendix.

Once an acceptance-rejection decision is made, the result is returned to the *MatchRun* function. If it is an acceptance, *MatchRun* stops traversing the offer list, returns the accepting patient's *id* and *DSA* to the *Allocate* function and thereupon to LivSim's *OrganArrival* call. If it is a rejection, *MatchRun* increments *noffers* and continues traversing the offer list or returns that it was not able to find a recipient to the *Allocate* function if the maximum number of offers were reached. Note that *noffers* is only incremented when *Offer* returns a negative decision, not when *MatchCheck* returns a negative decision – that is, offers are only counted as such if there was an opportunity for them to be rejected by the patient.

2.3.5 End-of-Year

End-of-Year is modeled through the *Year* function in event.py. This event occurs after each replication-year and instructs LivSim to calculate annual statistics, to write current annual statistics to the simulation output, and to initialize annual statistics for the following replication year.

2.3.6 End-of-Replication

End-of-Replication is modeled through the *EndRep* function. This event occurs after each replication. Currently, it only produces a time stamp of when the replication completed. This event is included for users wishing to add extra functionality or replication-dependent statistics to LivSim.

2.4 Modifying LivSim

This section provides tips for users wishing to modify LivSim. They are not meant to be comprehensive or taken as the only way to implement the modification, but they are the authors' opinions on the best course of action.

2.4.1 Modifying Geographic Structures

The advantage of LivSim is that geographic structures are modeled mathematically as binary relations (i.e. directed graphs) on the set of DSAs. This generality allows users to specify different regional systems, networks, topologies, etc. at the DSA/OPO level.

2.4.2 Adding New Events

This has to be done carefully. The first step is to create properly formatted input files containing the input streams. It is suggested to use a format similar to the existing input files. For example, the first columns

should contain the Replication number, DSA, and the event time followed by the event information. Second, *InputData_LivPlayback_1_11.py* must be modified to read to extra input file and if necessary, format it. Third, a pointer variable and index corresponding to the dimension the input file must be created prior to scheduling in the simulation engine. Third, the scheduling logic must be modified to advance the newly created pointer, determine the event time, determine whether this event precedes or follows other events, and pass the event information and event name to the calendar. This should be straightforward and closely resemble the code for the other events. Lastly, the user has to write a new function *User_Event()* or some other valid name that will be executed each time the corresponding event notice is retrieved from the calendar.

2.4.3 Adding New Donor or Patient Characteristics

Adding characteristics is simple. The user can just provide additional attributes to the *patient* or *organ* entity classes that correspond to the desired characteristics. Potential examples are transplant centers and donor hospitals. Although LivSim was first written and implemented at the DSA level, its extensibility allows transplant centers and donor hospitals to be modeled as characteristics of patients and donors. This can provide additional enrichment to the acceptance models, allocation rules, and geographic sharing structures.

2.4.4 Modifying Allocation Rules

If the user wishes to make any other changes to how the match run is created, changing how the offer list is compiled and sorted is also straightforward. The user can sort by another characteristic (instead of allocation MELD) and compile lists in different order (e.g. Share 35 national, followed by MELD 15-35 local, etc.) Additionally, if the user wishes to alter compatibility/cross-matching criteria, this can be accomplished by adding additional constraints to the *MatchCheck* function.

The *Offer* function is general. The user can replace it with a very general acceptance model (e.g. transplant-center specific), function, or even another decision simulation provided that it returns the final outcome of the acceptance/rejection decision.

2.4.5 Adding Additional Statistics

Adding additional statistics needs to be carefully done as well. First, the user will need to modify the *G* class, the global variable class object, and specify the dimensions of the output for the statistic. The dimensions should account for additional columns to make the output meaningful, such as the simulation replication and year. Second, if the statistics is be updated after each replication or replication-year, it needs to be added to the *SimStat* class. Third, the statistic, if again replication- or replication-year-dependent, needs to be initialized at the beginning of each replication when an instance of *SimStat* is created. Fourth, the code for calculating the statistic will be required. This step will vary based on the purpose and nature of the statistic, but code will likely have to be added to some event call (e.g. the number of patient arrivals is updated during the patient *Arrival* event). Fifth, if the statistic is replication-year or replication dependent, the statistic will have to be processed during the *End-of-Year* or *End-of-Replication* events respectively. Processing entails the formatting the statistic (e.g. adding headers and helpful columns such as the replication number, etc.); the writing of the statistic to the output; and clearing or re-initializing the statistic for the next replication or replication-year. If not replication-dependent, the statistic can be written directly to the output when calculated. Lastly, the user then needs to add code at the end of the simulation to write the output to the appropriate directory.

3 Postprocessing

The standard files for LivSim estimate OPTN behavior and waitlist outcomes. Analysis of transplant outcomes is done outside LivSim through post-processing of LivSim output using LSAM input files. Output delivered from post-processing includes:

1. The number of post-transplant mortalities by year.
2. Number of relist-waitlist mortalities and post-re-transplant mortalities by year.
3. Average organ transport distances, times, and mode of transport (ground vehicle, plane, or helicopter) by year.

The files used for each of these items are found in *PostTransplantEstimator_2.py*, *OutcomeEstimator_Relists_Regrafts_2.py*, and *Distance_Estimator_Run_2.py*, respectively, and are discussed in the following sections. These files are called altogether by the file *create_new_stats.py*. In order to run the code, one needs to do the following:

1. Mac or Linux:

- Open command terminal and change to the directory containing the post-processing files (ref. pg. 5).

Windows:

- Open Anaconda prompt and change to the directory containing the post-processing files (ref. pg. 5).

2. Type and enter the following:

python create_new_stats.py {D}

where

- D: Directory where the output files will be saved in.

Example:

python create_new_stats.py "C:/Users/JohnDoe/Documents/"

3.1 Post-Transplant Outcomes

The source code for calculating post-transplant outcomes is in *PostTransplantEstimator_2.py*. The user will need the following files from both LivSim and LSAM to run the code:

- LivSim Input Files:
  - status_times.txt
  - step_survival.txt
  - survivalcoefficients.txt
- LivSim Output Files:
  - RawOutput_TxID.csv
  - RawOutput_DoID.csv
- LSAM Files:
  - Input files for patients, status changes, and the initial waitlist.

The donor *id* and patient *id* attributes in both files must match. The modules will scan the LSAM files to retrieve appropriate donor and patient characteristics for the survival computations. After obtaining the necessary files, the user will have to modify the pathnames in *PostTransplantEstimator_2.py*. Afterwards, running the code will return the average number of post-transplant mortalities per year by replication.

Calculation of post-transplant survival in LivSim matches that of LSAM[9]. The survival model is a Cox model step function that determines patient survival according to

$$P(S > t) = f(t)^{\exp(\beta^T x)} \tag{3}$$

where S is the survival time, $f(t)$ is the baseline survival step function, x is the vector of covariates in the survival model, and β is the coefficient vector. The list of covariates and their coefficients are provided in Table 10. Consider a partition of the interval $[0, \bar{t})$ into n subintervals $[0, t_1), [t_1, t_2), ..., [t_{n-1}, \bar{t}]$ and let each subinterval (t_{i-1}, t_i) be associated with a survival probability v_i. The step function $f(t)$ returns v_k such that $t \in [t_{k-1}, t_k)$. Moreover, it is assumed that $v_k > v_{k+1}$, $v_0 = 1$, and $v_n = 0$. That is, $\{v_i\}_{i=1}^n \subset [0, 1]$ and it is strictly monotonically decreasing.

For each transplant event, the remaining survival time is calculated by sampling value u from a standard uniform distribution and inverting the complementary cumulative distribution function of S:

$$P(S > t) = u \implies t = \inf_k \left\{ t_k : \exp\left(\frac{\log(u)}{\beta^T x}\right) \in [v_k, v_{k-1}) \right\}. \tag{4}$$

Each replication performed will sample a different value of u and calculate the survival time t for that transplant recipient. If the sum of t and the current time of transplant is less than *maxtime*, death for the corresponding recipient during the time period is indicated.

3.2 Relist and Re-Transplant Outcomes

The source code for calculating relist and re-transplant outcomes is in *OutcomeEstimator_Relists_Regrafts_2.py*. The structure of the code is very similar to that of the calculation for post-transplant outcomes discussed in the previous section. The user will need the following files from both LivSim and LSAM to run the code:

- LivSim Input Files:
  - status_times.txt
  - stepsurvival.txt
  - survivalcoefficients.txt
- LivSim Output Files:
  - RawOutput_Relistid.csv
  - RawOutput_TxIDregraft.csv
  - Output_waitlistrelist_deaths.csv
  - RawOutput_DoIDregraft.csv
- LSAM Files
  - The input files for patients, status changes, and the initial waitlist.

The donor *id* and patient *id* attributes in both files must match. The modules will scan the LSAM files to retrieve appropriate donor and patient characteristics for the computations. After obtaining the necessary files, the user will have to modify the pathnames in *OutcomesEstimator_Relists_Regrafts_2.py*. Also, the user will have to input the probability that a relist candidate will die on the waitlist (*dprob*, default is about 15.2% based on OPTN data).

This code performs two calculations: the average number of mortalities for relist candidates and the average number of post-transplant mortalities for re-transplant candidates. The latter calculation proceeds exactly similarly to the post-transplant survival calculation discussed in the previous section. For the former calculation, any candidate that either was relisted (but not re-transplanted) or flagged by LivSim during the main run that he or she will eventually relist will be considered. The code will first determine whether the candidate was relisted (i.e. 1st graft failed before the end of the run or equivalently that the candidate's *RelistTxTime* value is less than maxtime). Next, using a uniformly distributed random number u, the code will indicate that the candidate's death if $u < dprob$. The computation will be repeated for the selected number of replications and the average number of mortalities and standard errors will be reported.

3.3 Distance-Related Outcomes

The source code for calculating transport times, distances, and mode is found in *Distance_Estimator_Run_2.py* . The user will need the following files from both LivSim and LSAM to run the code:

- LivSim Output Files:
  - RawOutput_DSAs.csv
- LSAM Files:
  - The input files for patients, status changes, the initial waitlist, and the distance/times file containing the transport distances/times/modes for each transplant center-donor hospital combination.

The donor *id* and patient *id* attributes in both files must match. The modules will scan the LSAM files to retrieve appropriate donor and patient characteristics for the survival computations.

The code will then process each transplant event that occurred during the main LivSim run. For each transplant event, it will retrieve a corresponding donor hospital-transplant center combination from the LSAM files. Users may also request that a random donor hospital-transplant center combination with the constraints that the donor hospital and transplant centers are located in the DSAs corresponding to the organ's and patient's DSA method respectively. Using a donor hospital-transplant center combination, the code will determine the transport time, distance, and mode (helicopter, drive, and airplane) for the transplant event. After processing all the transplant events, the code will output summary statistics for the average transport distance, average transport time, percentage for each mode of transport.

4 Summary Statistics

After running LivSim and generating the new postprocessing statistics, comparative analysis of several cases with modifications to sharing and geographic policies relative to a default case can be summarized through the files *compute_vol_diff.py*, *DSA_meld.py*, *mean_diff_summarize.py*, and *mean_summarize.py*. The following subsections will discuss what each file does.

4.1 Statistics Summary Table

A summary table of the raw number statistics can be produced by *mean_summarize.py*. The file requires a list of cases. The table outputted from the file lists the following raw numbers:

- Annualized Waitlist Deaths
- Annualized Waitlist Removals
- Annualized Waitlist Relist Deaths
- Annualized Waitlist Post-Transplant Deaths
- Annualized Post-ReTransplant Deaths
- DSA Mean Transplant MELD
- Standard Deviation of DSA Mean Transplant MELD
- DSA Median Transplant MELD
- Standard Deviation of DSA Median Transplant MELD
- Average Organ Transport Distance (Ground Vehicle)
- Average Organ Transport Distance (Helicopter)
- Average Organ Transport Distance (Airplane)
- Average Organ Transport Time (Ground Vehicle)
- Average Organ Transport Time (Helicopter)
- Average Organ Transport Time (Airplane)
- Average Percentage of Organ Transported by Ground Vehicle
- Average Percentage of Organ Transported by Helicopter
- Average Percentage of Organ Transported by Airplane

In addition, when one wants to compare a list of cases to a base or default case, one can use *mean_diff_summarize.py* to compute the differences of the raw numbers between a tested case and a base case. Along with those differences, *mean_diff_summarize.py* performs a t-test for each difference and computes its corresponding p-value to assist in evaluating the significance of the difference.

Before running *mean_summarize.py* and *mean_diff_summarize.py*, one needs to specify the output directory to save the summary tables. The user can do so by changing the variable *output_directory* in both python files. For *mean_diff_summarize.py*, to specify a base case to compare against, the user needs to change the value of *base_directory* in the code so that the code can access the output files for the base case. To specify the list of directories corresponding to the cases that the user wants to analyze, the user needs to list them in the *files* variable in the code. Moreover, the user needs to name each of the cases through the *cases* variable. Once these steps are complete, the user can run *mean_summarize.py* and *mean_diff_summarize.py* as scripts and their corresponding summary tables will be generated.

4.2 DSA-Level Organ Volume Gain/Loss

The file *compute_vol_diff.py* computes the relative difference with respect to the default case for each DSA. It not only outputs a table listing the volume percentage gain/loss for each DSA, but it also creates a table listing the minimum, the maximum, and the quartiles of the relative difference for each case.

The file *compute_vol_diff.py* requires each case's corresponding RawOutput_DSAs.csv file. Like in *summarize.py*, one needs to specify the directory of the output files for the default case through the variable *base_directory* and the lists of directories corresponding to the other cases through *files*. The user needs to name each of the case through the list *cases*. Finally, to save the table in the desired directory, the user needs to specify it through the variable *output_directory*. Once these steps are complete, the user can run *compute_vol_diff.py* as a script and the tables will be generated.

4.3 DSA-Level Mean MELD

The file *DSA_meld.py* computes the mean MELD for each DSA. It outputs a table listing the mean MELD for each DSA for each case.

The file *DSA_MELD.py* requires each case's corresponding RawOutput_ytransplnats.csv and RawOutput_yMELD.csv. Like in *mean_diff_summarize.py*, one needs to specify the directory of the output files for the default case through the variable *base_directory* and the lists of directories corresponding to the other cases through *files*. The user needs to name each of the case through the list *cases*. Finally, to save the table in the desired directory, the user needs to specify it through the variable *output_directory*. Once these steps are complete, the user can run *DSA_meld.py* as a script and the tables will be generated.

5 Appendix

5.1 Reduced Form Acceptance Model

This alternative acceptance model and code may be used to avoid using LSAM inputs. The acceptance model uses LSAM's coefficients for whether the potential recipient is Status 1, the potential recipient's waiting time, whether the potential recipient is listed in the DSA of the procuring OPO, and donor blood type and assumes all other patient attributes are held at the baseline. These four sets of coefficients included are also the four most significant predictors in LSAM's acceptance model.

Characteristic	Coefficient
Constant	-2.88843
Status 1 = True	1.0889
Local Transplant = True	0.7619
Donor Blood Type = AB	0.75528
Donor Blood Type = B	0.6677
Waiting Time (years)	-0.08531145

Table 36: Format of Input_Acceptance.txt Data

```
def Offer(offered_organ, matching_recipient):
#This function offers an organ to a patient and
#returns information based on acceptance/rejection accept =1


    #Generate acceptance decision
    r1 = nump.random.uniform(0,1,1)

    #Characteristcs
    patientx = [1, matching_recipient.Status1,
    int(matching_recipient.DSA == offered_organ.DSA),int(1== offered_organ.ABO),
    int(2== offered_organ.ABO),(Sim.clock-matching_recipient.create_time)]

    accept_prob =
    nump.exp(nump.dot(patientx,AcceptanceModel)) / (1+
    nump.exp(nump.dot(patientx,AcceptanceModel)))
```

```
#accept_prob = .05
accept = int(r1 <= accept_prob)

#Return information based on decision
if accept ==1:
    return [1,0,matching_recipient.DSA, matching_recipient.id]

else:
    return [0,1,[],[]]
```

References

[1] Organ procurement and transplantation network data report. `https://optn.transplant.hrsa.gov/data/`. [Online; accessed 21-July-2015].

[2] Policy proposals: proposal to add serum sodium to the meld score, 2016.

[3] Revised liver policy regarding hcc exception scores. `https://optn.transplant.hrsa.gov/news/revised-liver-policy-regarding-hcc-exception-scores/`, 2016. [Online; accessed July-2016].

[4] Saleh Elwir and John Lake. Current status of liver allocation in the united states. *Gastroenterology & hepatology*, 12(3):166, 2016.

[5] Aditya K Iyer, Andrew J Schaefer, Cindy L Bryce, Gabriel L Zenarosa, Chung-Chou H Chang, and Mark S Roberts. A biologically based discrete-event simulation model of liver transplantation in the united states for pediatric and adult patients. In *Proceedings of the Winter Simulation Conference*, pages 1275–1282. Winter Simulation Conference, 2011.

[6] Vikram Kilambi and Sanjay Mehrotra. Improving liver allocation using optimized neighborhoods. *Transplantation*, 101(2):350–359, 2017.

[7] Jennifer Kreke, Andrew J Schaefer, Derek C Angus, Cindy L Bryce, and Mark S Roberts. Methods for special applications: incorporating biology into discrete event simulation models of organ allocation. In *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*, pages 532–536. Winter Simulation Conference, 2002.

[8] A Alan B Pritsker, David L Martin, Janet S Reust, Mary Ann Wagner, O Patrick Daily, Ann M Harper, Erick B Edwards, Leah E Bennett, James R Wilson, Michael E Kuhl, et al. Organ transplantation policy evaluation. In *Proceedings of the 27th conference on Winter simulation*, pages 1314–1323. IEEE Computer Society, 1995.

[9] Scientific Registry for Transplant Recipients. *Liver Simulated Allocation Model:User's Guide*, 2014.

[10] Steven M Shechter, Cindy L Bryce, Oguzhan Alagoz, Jennifer E Kreke, James E Stahl, Andrew J Schaefer, Derek C Angus, and Mark S Roberts. A clinically based discrete-event simulation of end-stage liver disease and the organ allocation process. *Medical Decision Making*, 25(2):199–209, 2005.

[11] Sarah E Taranto, Ann M Harper, Erick B Edwards, John D Rosendale, Maureen A McBride, O Patrick Daily, Dan Murphy, Bill Poos, Janet Reust, and Bruce Schmeiser. Developing a national allocation model for cadaveric kidneys. In *Simulation Conference, 2000. Proceedings. Winter*, volume 2, pages 1971–1977. IEEE, 2000.

[12] James F Trotter. Current issues in liver transplantation. *Gastroenterology & hepatology*, 12(4):214, 2016.

[13] Stefanos A Zenios, Glenn M Chertow, and Lawrence M Wein. Dynamic allocation of kidneys to candidates on the transplant waiting list. *Operations Research*, 48(4):549–569, 2000.