Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерства освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерства освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

САРНАЦЬКИЙ ВЛАДИСЛАВ ВІТАЛІЙОВИЧ

УДК: 004.434, 004.942

ДИСЕРТАЦІЯ

МОВА ПРОГРАМУВАННЯ ТА ПРОГРАМНІ ЗАСОБИ ОПИСУ АГЕНТНИХ МОДЕЛЕЙ РОЗПОВСЮДЖЕННЯ ІНФЕКЦІЙНИХ ЗАХВОРЮВАНЬ

121 Інженерія програмного забезпечення

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ В.В. Сарнацький

Науковий керівник: Баклан Ігор Всеволодович, кандидат технічних наук, доцент

Київ - 2023

АНОТАЦІЯ

Сарнацький В.В. Мова програмування та програмні засоби опису агентних моделей розповсюдження інфекційних захворювань. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 – Інженерія програмного забезпечення та 12 – Інформаційні технології. – Національний Технічний Університет України «Київський Політехнічний Інститут імені Ігоря Сікорського», Київ, 2023.

Дисертаційна робота присвячена розробці інструментального забезпечення для ефективного моделювання процесів розповсюдження інфекційних захворювань.

Висвітлено основні напрями та тенденції розвитку сфери епідеміологічного моделювання у цілому та зокрема процесу розробки агентних епідеміологічних моделей. Були виявлені переваги та недоліки запропонованих дослідницькою спільнотою рішень та, зокрема, обсяг, гнучкість та швидкодія програмних засобів моделювання. Серед програмних засобів моделювання не було помічено представників, що поєднують у собі доступний інтерфейс з високою швидкодією та гнучкістю опису моделей. Був зроблений висновок, що розробка предметно-орієнтованої мови програмування для опису агентних моделей та графічного середовища розробки та аналізу нададуть доступний інтерфейс опису моделі у поєднанні зі значною гнучкістю та швидкодією.

Виконана побудова загального математичного апарату епідеміологічної моделі агентного типу. У рамках математичної моделі була сформульована основна складність агентного епідеміологічного моделювання, а саме урахування контактів під час обчислення розподілу майбутнього стану захворювання. Запропоновано три алгоритми обчислення цього розподілу, проаналізована їх алгоритмічна складність. Також, для одного з них – алгоритму з групуванням було виконане дослідження оптимальної його імплементації.

З огляду на розроблену математичну модель було дано визначення формальної граматики предметно-орієнтованої мови опису агентної епідеміологічної моделі CTrace. Мова надає вбудований високорівневий інтерфейс для задання основних сутностей заданої загальної епідеміологічної моделі, а також відношень між ними. Мова підтримує задання ймовірнісних розподілів різного типу та основні операції між ними що значно спрощує задання соціодемографічної параметризації моделей.

Виконана побудова транслятора розробленої формальної мови опису агентної епідеміологічної моделі. Розроблений транслятор є компілятором із конфігурованою проміжною мовою. Такою мовою може бути Python, який своєю чергою транслюється у мову машинних інструкцій під час процесу JIT-компіляції, або будь-яка мова загального призначення, що має інтерфейс до побудови Python-модулів. Такою мовою була обрана мова Rust. Основним аспектом алгоритму трансляції є мінімізація кількості виділень пам'яті у процесі виконання результівної програми. Для цього, компілятор обраховує загальний обсяг пам'яті, необхідний для роботи моделі, і використовує буфер відповідного розміру для організації усіх обчислень.

Розроблене середовище розробки та аналізу агентних епідеміологічних моделей з використанням мови CTrace – CTraceEnv. Розроблене середовище надає базовий функціонал необхідний для підтримки процесу опису моделей мовою CTrace, що включає підсвітку синтаксису вихідного коду, вбудований транслятор тощо. Серед функціоналу аналізу представлені елементи керування роботою моделі, значень її глобальних параметрів, інтерфейс перегляду динаміки розповсюдження досліджуваного інфекційного захворювання, а також функціонал експорту результатів для подальшого аналізу сторонніми спеціалізованими інструментами.

Проведений аналіз розробленої мови опису агентних епідеміологічних моделей з боку доцільності її використання, обсягу сценаріїв, що можуть бути

змодельовані, ефективності розробки та використання

Виконана практична апробація розробленого інструментарію для ефективного моделювання розповсюдження інфекційних захворювань. У її рамках, було дано визначення агентної епідеміологічної моделі розповсюдження коронавірусу SARS-CoV-2 серед населення Польщі у період з початку вересня 2020 року до кінця листопада 2020 року, що була відкалібрована з використанням публічно доступних епідеміологічних даних. Результівна модель описує динаміку зміни кількості інфікованих людей з коефіцієнтом детермінації рівним 0.9319, що свідчить про здатність розробленого підходу описувати процеси розповсюдження інфекційних захворювань.

Проведений аналіз впливу соціодемографічної гетерогенності середовища моделювання на якість карантинних стратегій побудованих на його основі. Відповідний експеримент полягав у реалізації алгоритму пошуку карантинних стратегій для заданої агентної епідеміологічної моделі країни Європейського Союзу. Соціодемографічна параметризація моделі побудована з використанням статистичного моделювання на основі відкритих історичних даних, з відкритої бази статистичних даних Eurostat. Знайдені алгоритмом стратегії були порівняні із тривіальними, що показало більшу їх ефективність та використані у процесі перехресного порівняння. Для цього, стратегія, отримана з моделі країни тестування була порівняна з іншими. В обох випадках критерієм порівняння виступала сумарна винагорода за дії агента, що використовує відповідну стратегію у середовищі моделювання, сформульованим як Марківський процес вирішення. Аналіз показав, що для покращення їх якості соціодемографічні показники мають бути враховані, що, своєю чергою, показує необхідність наявності їх задання у будь-якому інструменті епідеміологічного моделювання. Це свідчить про доцільність тої уваги, що була приділена процесу їх задання при розробці мови CTrace.

Окремо був проведений якісний аналіз спектра сценаріїв, що можуть бути

змодельовані з використанням мови CTrace. Показано моделювання таких аспектів агентного епідеміологічного моделювання як: динамічні параметри моделі та її сутностей, географія середовища, погодні умови, векторні захворювання.

Аналіз ефективності розробки епідеміологічних моделей мовою CTrace показав значне скорочення обсягу вихідного коду програми у порівнянні з моделями імплементованими мовами програмування загального призначення. В якості цих моделей, був використаний лістинг програмного коду проміжною мовою, що генерується транслятором. Скорочення обсягу програмного коду становило від 14 до 34 разів для мови Python та від 16 до 45 разів для мови Rust. Це скорочення може свідчити про зменшення кількості людино-годин необхідних для їх розробки, а також збільшену швидкість ітерування. Окрім цього, зменшення обсягу вихідного коду призводить до зменшення кількості помилок, що, своєю чергою, зменшує час опису моделей.

Дано визначення метрики функціональності інструменту епідеміологічного моделювання, що дозволяє оцінити обсяг сценаріїв та функцій доступних користувачу.

Дано визначення метрики ефективності моделі, що дозволяє виконувати порівняння різних моделей та інструментів моделювання з точки зору їх обчислювальної ефективності. Ця метрика є інваріантною до кількості агентів, тривалості симуляції, її гранульованості та кількості одночасно залучених до обчислення моделі ядер центрального процесору.

Аналіз швидкодії результівних моделей, заданих мовою CTrace, ставив дві мети: обґрунтувати доцільність використання мови Python як проміжної мови трансляції, а також порівняти їх швидкодію з наявними моделями. Показано, що використання мови Python з бібліотекою для JIT-компіляції Numba дозволяє отримати швидкодію моделей, порівняну з отриманою у разі використання мови Rust, ефективність якої порівняна з C/C++. При цьому, гнучкість мови Python дозволяє відносно просто вносити корективи у процес трансляції. Екс-

перименти з використанням різних тестових середовищ показали, що існуючі епідеміологічні моделі не виграють у швидкодії у розроблених мовою CTrace еквівалентів.

З результатів порівняння розробленого середовища CTraceEnv з аналогами за визначеними метриками ефективності та функціональності можна зробити висновок що обсяг функцій агентного епідеміологічного моделювання наданий CTraceEnv є значно більшим за аналоги й приблизно відповідає середовищу NetLogo, надаючи при цьому ефективність результівних моделей, що не програє аналогам.

Ключові слова: моделювання; моделювання поведінки; індуктивне моделювання; лінгвістичне моделювання; симулювання; нейронна мережа; епідеміологія; математична модель; агентна модель; мультиагентна система; формальна мова; мова програмування; мова опису моделі; алгоритм; ітеративний алгоритм; середовище моделювання; організація обчислювальних ресурсів; аналіз ефективності; оптимізація; програмне забезпечення; архітектура програмного забезпечення; CTrace; CTraceEnv.

Список публікацій здобувача

Публікації у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б»:

1. Сарнацький В. В., Баклан І. В. МЕТОДИ ТА ЗАСОБИ МОДЕЛЮВАННЯ РОЗПОВСЮДЖЕННЯ ІНФЕКЦІЙНИХ ЗАХВОРЮВАНЬ. Вчені записки Таврійського національного університету імені В. І. Вернадського. т. 33. No 2. р. 2022. с 100-107;
2. Сарнацький В. В., Баклан І. В. ВПЛИВ СОЦІОДЕМОГРАФІЧНОЇ ГЕТЕРОГЕННОСТІ НА ОПТИМАЛЬНУ СТРАТЕГІЮ ВПРОВАДЖЕННЯ КАРАНТИННИХ ЗАХОДІВ. Вчені записки Таврійського національного університету імені В. І. Вернадського. т. 32. No 4. р. 2021. с 150-155;
3. Sarnatskyi V., Baklan I. CTraceEnv: A platform for development and analysis of agent-based epidemiological models using CTrace language. Вчені записки Таврійського національного університету імені В. І. Вернадського. т. 34. No 2. р. 2023. с. 100-107.

Публікації у наукових виданнях, включених до наукометричної бази Scopus:

4. Sarnatskyi V., Baklan I. On Efficient Single-Core Execution of Agent-Based Epidemiological Models with Contact-Tracing Transmission. Proceedings of The Sixth International Workshop on Computer Modeling and Intelligent Systems (CMIS 2023), vol.3392, 2023. – с. 23-36;
5. Sarnatskyi V., Baklan I. CTrace: Language for Definition of Epidemiological Models with Contact-Tracing Transmission. Lecture Notes on Data Engineering and Communications Technologies, vol.149 – Springer, Cham, 2023. – С. 426-448.

Публікації у матеріалах наукових конференцій:

6. Сарнацький, В. В. Мовний засіб опису агентних епідеміологічних моделей / Сарнацький Владислав Віталійович, Баклан Ігор Всеволодович // Інженерія програмного забезпечення і передові інформаційні технології (SoftTech-2022):

матеріали II та III Всеукраїнських науково-практичних конференцій молодих вчених та студентів, присвячених 125-й річниці КПІ ім. Ігоря Сікорського (22–26 травня та 23-25 листопада 2022 р., Київ). – Київ : КПІ ім. Ігоря Сікорського, ІПІ ФІОТ, 2022. – С. 85-89. – Бібліогр.: 17 назв.

ABSTRACT

Sarnatskyi V.V. Programming Language and Software Tools for Description of Agent-Based Models of the Spread of Infectious Diseases. – Qualified scientific work on the rights of the manuscript. Dissertation for the degree of Doctor of Philosophy in the specialty 121 - Software Engineering and 12 - Information Technology. - National Technical University of Ukraine ”Igor Sikorsky Kyiv Polytechnic Institute Kyiv, 2023.

The dissertation is devoted to the development of tools for effective modeling of infectious disease spread.

The main directions and trends in the development of epidemiological modeling in general and the process of developing agent-based epidemiological models in particular are highlighted. The advantages and disadvantages of the solutions proposed by the research community and, in particular, the scope, flexibility and speed of modeling software were identified. Among the modeling software tools, no representa-tives were found that combine an accessible interface with high performance and flexibility of model description. It was concluded that the development of a domain-specific programming language for describing agent-based models and a graphical development and analysis environment will provide an accessible interface for describing the model in combination with significant flexibility and performance.

The general mathematical apparatus of an agent-based epidemiological model was developed. Within the framework of the mathematical model, the main difficulty of agent-based epidemiological modeling was formulated, namely, taking into account contacts when calculating the distribution of the future state of the disease. Three algorithms for calculating this distribution were proposed, and their algorithmic complexity was analyzed. Also, for one of them, the algorithm with grouping, a study of its optimal implementation was performed.

Given the developed mathematical model, the formal grammar of the domain-

specific language for describing the CTrace agent-based epidemiological model was defined. The language provides a built-in high-level interface for specifying the main entities of a given general epidemiological model, as well as the relationships between them. The language supports the definition of probability distributions of various types and basic operations between them, which greatly simplifies the sociodemographic parameterization of models.

The translator of the developed formal language for describing the agent-based epidemiological model was built. The developed translator is a compiler with a configurable intermediate language. Such a language can be Python, which in turn is translated into a machine instruction language during the JIT compilation process, or any general-purpose language that has an interface to building Python modules. Rust was chosen as such a language. The main aspect of the translation algorithm is to minimize the number of memory allocations during the execution of the resulting program. To do this, the compiler calculates the total amount of memory required for the model and uses a buffer of the appropriate size to organize all calculations.

An environment for the development and analysis of agent-based epidemiological models using the CTrace language – CTraceEnv – was developed. The developed environment provides the basic functionality necessary to support the process of describing models in CTrace, including source code syntax highlighting, a built-in translator, etc. The analysis functionality includes controls for model operation, values of its global parameters, an interface for viewing the dynamics of the spread of the studied infectious disease, as well as the functionality of exporting results for further analysis by third-party specialized tools.

The developed language for describing agent-based epidemiological models was analyzed in terms of the feasibility of its use, the scope of scenarios that can be modeled, and the efficiency of its development and use.

The developed tools for effective modeling of the spread of infectious diseases were practically tested. Within its framework, an agent-based epidemiological model

of the spread of the SARS-CoV-2 coronavirus among the population of Poland in the period from early September 2020 to the end of November 2020 was defined, which was calibrated using publicly available epidemiological data. The resulting model describes the dynamics of changes in the number of infected people with a coefficient of determination of 0.9319, which indicates the ability of the developed approach to describe the processes of spreading infectious diseases.

The impact of sociodemographic heterogeneity of the modeling environment on the quality of quarantine strategies based on it was analyzed. The corresponding experiment consisted of implementing an algorithm for searching for quarantine strategies for a given agent-based epidemiological model of a European Union country. The sociodemographic parameterization of the model was built using statistical modeling based on open historical data from the open database of Eurostat. The strategies found by the algorithm were compared with the trivial ones, which showed their greater efficiency and were used in the crosscomparison process. For this purpose, the strategy obtained from the model of the test country was compared with the others. In both cases, the comparison criterion was the total reward for the actions of the agent using the corresponding strategy in the modeling environment, formulated as a Markov decision process. The analysis showed that to improve their quality, sociodemographic indicators should be taken into account, which, in turn, shows the need for their setting in any epidemiological modeling tool. This demonstrates the appropriateness of the attention paid to the process of their specification in the development of the CTrace language.

Separately, a qualitative analysis of the range of scenarios that can be modeled using the CTrace language was conducted. It was shown how to model such aspects of agent-based epidemiological modeling as: dynamic parameters of the model and its entities, geography of the environment, weather conditions, vector-borne diseases.

The analysis of the efficiency of developing epidemiological models in CTrace showed a significant reduction in the amount of program source code compared to

models implemented in general-purpose programming languages. As these models, the program code listing in an intermediate language generated by the translator was used. The reduction in program code size ranged from 14 to 34 times for Python and from 16 to 45 times for Rust. This reduction may indicate a decrease in the number of man-hours required for their development, as well as an increased iteration rate.

A definition of the functionality metric of an epidemiological modeling tool was given.

The model efficiency metric was defined, which allows comparing different models and modeling tools in terms of their computational efficiency. This metric is invariant to the number of agents, the duration of the simulation, its granularity, and the number of CPU cores simultaneously involved in the model calculation.

The analysis of the performance of the resultant models defined in CTrace had two goals: to justify the feasibility of using Python as an intermediate translation language, and to compare their performance with existing models. It has been shown that using Python with the Numba JIT compilation library allows to obtain model performance comparable to that obtained by using Rust, whose efficiency is comparable to C/C++. At the same time, the flexibility of Python makes it relatively easy to make adjustments to the translation process. Experiments using various test environments have shown that existing epidemiological models do not outperform the equivalents developed in CTrace.

Based on the results of comparing the developed CTraceEnv environment with analogues by certain efficiency and functionality metrics, it can be concluded that the scope of agent-based epidemiological modeling functions provided by CTraceEnv is much larger than analogues and approximately corresponds to the NetLogo environment, while providing the efficiency of the resulting models that does not lose to analogues.

Keywords: modeling; behavioral modeling; inductive modeling; linguistic modeling; simulation; neural network; epidemiology; mathematical model; agent-

based model; multi-agent system; formal language; programming language; model description language; algorithm; iterative algorithm; simulation environment; computational resource management; performance analysis; optimization; software; software architecture; CTrace; CTraceEnv.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

• SIS, SIR, SIRS, SEIR, SEIS, SEIRS, SEI_aI_sRS - типи компартментних епідеміологічних моделей;

• ABM, IBM (agent-based model, individual-based model) - агентна модель;

• F^2-QMRA - агентна модель розповсюдження мікроорганізмів у межах закладів харчування;

• FluTE - програмний компонент для побудови агентних епідеміологічних моделей;

• GLEaM (Global Epidemic and Mobility) - програмний компонент для побудови компартментних епідеміологічних моделей;

• GLEaMvis - середовище аналітики GLEaM;

• STEM (Spatiotemporal Epidemiological Modeler) - програмний компонент для епідеміологічного моделювання;

• FRED (Framework for Reconstructing Epidemic Dynamics) - програмний компонент для побудови агентних епідеміологічних моделей;

• GLSL (OpenGL Shading Language, Graphics Library Shader Language) - мова опису шейдерів;

• СУБД (Система Управління Базами Даних) - набір взаємопов’язаних даних (база даних) і програм для доступу до цих даних;

• JIT-компіляція (Just-In-Time компіляція) - технологія пришвидшення виконання програмного коду шляхом його компіляції безпосередньо під час виконання;

ВСТУП

Актуальність теми. Історія людства налічує численні випадки як локальних, так і глобальних епідемій. Перші задокументовані з них сягають часів XIV сторіччя до нашої ери. Так, із глиняних табличок Амарнського архіву, знайдених на території стародавнього Єгипту [1], стало відомо про ”чуму з Мегіддо”. Зі слів правителя цього міста, регіон був ”поглинутий смертю, чумою та пилом”. І хоча збудник цієї хвороби залишається невідомим, як і кількість потерпілих, цей випадок став одним із перших задокументованих проявів захворювання у масштабах епідемії.

Розпочавшись близько двох з половиною тисяч років після, у XIV сторіччі, друга епідемія бубонної чуми (після чуми Юстиніана у VI сторіччі), широко відома під назвою “чорна смерть”, за різними оцінками стала причиною смерті від 75 до 200 мільйонів людей, що становило 17-54% всього населення світу [2, 3]. Наперекір катастрофічним людським втратам, та негативному економічному впливу у вигляді скорочення обсягів робочої сили та відповідного підвищення податків [4], ця епідемія дала значний товчок для розвитку медицини в області дослідження механізмів розповсюдження захворювань. І хоча медицина мала в основі вчення стародавніх греків про чотириелементну будову Всесвіту, один з елементів — повітря, був вірно окреслений як переносник захворювання. Звісно, людство у XIV не мало уяву про присутні у цьому повітрі мікроорганізми, але процес провітрювання приміщень із хворими став широкою практикою [5].

Попри досить глибоке розуміння механізмів розповсюдження інфекційних захворювань та високий рівень медицини у цілому, притаманні сучасному людству, досвід пандемії COVID-19, спричиненою коронавірусом SARS-CoV-2 [6] показав повну непідготовленість людства до таких глобальних епідемій. Те, що розпочиналось як місцеве захворювання серед мешканців міста Ухань провінції Хубей у грудні 2019 року [7], було класифіковано Всесвітньою організацією

охорони здоров’я як пандемія вже у березні 2020 року [8]. Станом на квітень 2022, ця пандемія лишила життя близько 6 мільйонів осіб та завдавала значного удару світовій економіці [9]. Так, кошти на підтримку системи охорони здоров’я країн з низьким та середнім рівнем доходів населення сягають приблизно 52 млрд $ (8.60$ на людину) кожні чотири тижні [10]. За цією ж оцінкою Сполучені Штати Америки витрачають на це 50.7 млрд $ кожен місяць. Окрім економічного удару, є свідчення про значний негативний вплив і на психологічне здоров’я населення [11, 12].

Попри значну кількість різних методів моделювання розповсюдження інфекційних захворювань, незмінним є наступне твердження: своєчасне, стратегічно обумовлене впровадження карантинних та інших засобів протидії поширенню інфекційного захворювання є найефективнішим методом боротьби з епідемією. Таке впровадження сприяє ”згладжуванню”кривої захворюваності, що дозволяє ефективно розпоряджатись медичними ресурсами (медичне обладнання, препарати, місця у лікарнях) та запобігає їх дефіциту, що спостерігався під час пандемії COVID-19 [13, 14, 15].

Пошук та аналіз стратегій боротьби з розповсюдженням інфекційних захворювань не є можливим без їх використання в умовах контрольованого експерименту у середовищі моделювання. Наявність точних моделей та їх соціодемографічна параметризація має значний вплив на якість оптимальних стратегій у питанні застосування щодо реального світу [16]. В останні роки спостерігається значний ріст долі досліджень спрямованих на використання агентно-орієнтованих моделей для епідеміологічного моделювання. Цей ріст пояснюється більшою точністю у порівнянні з “класичними” компартментними моделями та зростанням доступних обчислювальних ресурсів, необхідних для їх обрахунку. Розробка таких моделей потребує від дослідника не тільки знань у сфері епідеміологічного моделювання, а і просунутих навичок програмування. Така потреба виникає через те, що агентно-орієнтоване моделювання роз-

глядає середовище як сукупність окремих агентів, кількість яких може сягати значних величин. Так, модель країни може складатись з мільйонів агентів, поведінка яких, має бути обрахована окремо. Для зниження необхідного рівня навичок програмування можуть бути використані програмні засоби у вигляді бібліотек для певної мови програмування або окремі середовища моделювання з явним або неявним використанням внутрішньої мови опису моделей. У разі використання останніх, мова опису моделі може бути спроєктована з ціллю максимального спрощення використання, що дозволить пришвидшити розробку нових моделей та їх аналіз.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота тісно пов'язана з науковими розробками, що здійснюються на кафедрі автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського». Тема дисертації відповідає науковому напряму "Інтелектуальні методи програмування, моделювання і прогнозування з використанням ймовірнісного і лінгвістичних підходів". Державний реєстраційний номер 0117U000926.

Мета і завдання дослідження. Метою дисертаційної роботи є створення інструментального забезпечення для ефективного моделювання процесів розповсюдження інфекційних захворювань.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- ознайомлення з наявним інструментарієм, що допомагає у створенні моделей процесів розповсюдження інфекційних захворювань;
- створення предметно-орієнтованої мови для ефективного моделювання процесів розповсюдження інфекційних захворювань;
- реалізація транслятора розробленої предметно-орієнтованої мови та інтегрованого середовища розробки з її використанням;
- практична апробація створеного інструментарію для ефективного моделю-

вання процесів розповсюдження інфекційних захворювань;

• порівняльний аналіз розробленого інструментарію з наявними аналогами.

Об'єкт дослідження – сучасні підходи до створення мов опису моделей.

Предмет дослідження – програмні інструменти опису агентних моделей.

Методи досліджень включають у себе систематичний пошук та аналіз літератури у сфері епідеміологічного моделювання. В процесі дослідження були використані методи статистичного аналізу, побудови граматики формальних мов, алгоритми лексичного та синтаксичного аналізу.

Наукова новизна одержаних результатів.

• *Вперше* дано визначення предметно-орієнтованої мови опису агентних моделей розповсюдження інфекційних захворювань, що, на відміну від мов опису моделей загального призначення, надає можливість гнучкого їх опису. Використання визначеної мови дозволило скоротити обсяг вихідного коду моделей у 14-45 разів у порівнянні з використанням мов програмування загального призначення.

• *Вперше* розроблений транслятор предметно-орієнтованої мови опису агентних епідеміологічних моделей. Використання розробленого транслятора для компіляції вихідного коду опису моделей призводить до створення ефективних агентних моделей розповсюдження інфекційних захворювань, час обчислення яких програє на 20% лише одному з наявних методів агентного епідеміологічного моделювання. У порівнянні з іншими, виграш у швидкості обчислення кроку моделі сягає від 2.9 до 1.6×10^4 разів

• *Набула подальшого розвитку* загальна агентна математична модель розповсюдження інфекційних захворювань, що, на відміну від існуючих, здатна ураховувати більшість сценаріїв характерних агентним епідеміологічним моделям.

Практичне значення одержаних результатів. Одержані результати дозволяють використовувати розроблену предметно-орієнтовану мову та програмне

середовище для прототипування, розробки та аналізу агентно-орієнтованих моделей розповсюдження інфекційних захворювань. Наявний програмний інтерфейс дозволяє виконувати інтегрування розроблених моделей в існуючі програмні системи.

Особистий внесок здобувача. Дисертація є результатом самостійних наукових досліджень, в яких викладено авторський підхід до побудови математичної моделі розповсюдження інфекційних захворювань шляхом трасування контактів, відповідної предметно-орієнтованої мови опису агентної моделі та середовища проєктування та аналізу. Усі наукові положення, отримані результати, описані у дисертації, були отримані здобувачем у самостійно у процесі науково-дослідчої роботи. В роботах, опублікованих у співавторстві, дисертанту належать: [17] - обґрунтований вплив соціодемографічної параметризації на ефективність карантинних заходів;

Апробація результатів дисертації. Результати роботи були представлені та опубліковані у рамках міжнародних та всеукраїнських конференцій а саме на: III Всеукраїнській науково-практичній конференції молодих вчених та студентів “Інженерія програмного забезпечення і передові інформаційні технології” (SoftTech-2022).

Публікації. За результатами дисертаційних досліджень опубліковано п’ять наукових праць, в тому числі три статті у наукових фахових виданнях України. Публікації входять до наступних наукометричних баз даних з міжнародним індексом цитування: Scopus – дві.

Структура і обсяг роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел із 101 найменування та додатків. Загальний обсяг дисертації становить 173 сторінок, з яких 111 сторінок основного тексту, 4 додатки на 44 сторінках, та містить 48 рисунків, 31 формулу, 3 таблиці.

РОЗДІЛ 1.

МАТЕМАТИЧНІ МОДЕЛІ, МОВНІ ТА ПРОГРАМНІ ЗАСОБИ ДЛЯ МОДЕЛЮВАННЯ РОЗПОВСЮДЖЕННЯ ІНФЕКЦІЙНИХ ЗАХВОРЮВАНЬ

1.1 Компартментні моделі

Питанням побудови математичної моделі процесу розповсюдження епідемії людство задалось ще на початку XX століття. Так, у 1927 році була опублікована робота за авторством Андерсона МакКендрика та Уільяма Кермака, що узагальнює результати їх дослідження щодо побудови такої моделі [18]. Авторами була розглянута модель, яку надалі зарахують до класу компартментних. Згідно з моделями цього типу, процес протікання хвороби у межах одного індивіду відокремлюється на окремі стани, а суспільство на відповідні групи, кожну з яких описує функція частини модельованого суспільства у ній від часу. Ці функції можуть бути пов'язані одна з одною диференційними рівняннями, розв'язання системи яких є метою процесу моделювання.

Запропонована МакКендриком та Кермаком модель передбачає наявність трьох груп:

• Susceptible (S) - група, члени якої не були інфіковані та можуть бути інфікованими у майбутньому;

• Infectious (I) - група, члени якої були інфіковані та можуть інфікувати представників групи S;

• Recovered/Removed (R) - група, члени якої перенесли модельоване захворювання, не можуть бути інфікованими повторно та не можуть інфікувати інших. Ця група представляє частину населення, що придбала постійний імунітет, або померла.

Ця модель, як і її наступники, має назву утворену першими літерами назв цих груп - SIR.

Відповідні до цих груп функції долі населення у цих групах - $S(t), I(t), R(t)$ пов'язані наступною системою диференційних рівнянь:

$$\begin{cases} S'(t) = -\beta S(t)I(t) \\ I'(t) = \beta S(t)I(t) - \gamma I(t) \\ R'(t) = \gamma I(t) \end{cases}, \tag{1.1}$$

де β - темп розповсюдження, що показує середню кількість інфікувань кожним індивідуумом щодня; γ - темп відновлення, тобто доля інфікованих, що одужує щодня. Слід зазначити, що деякі автори використовують формулювання компартментних моделей, згідно з якими кожна з функцій показує кількість членів у цій групі, а не долю від популяції. Ці два формулювання є тотожними.

На рисунку 1.1 зображені графіки розв'язання задачі Коші цієї системи при: $\beta = 0.5, \gamma = \frac{1}{14}$ з початковими умовами: $S(0) = 0.999$, $I(0) = 0.001$, $R(0) = 0$. Корені були знайдені за допомогою методу Рунге-Кутти четвертого порядку [19].

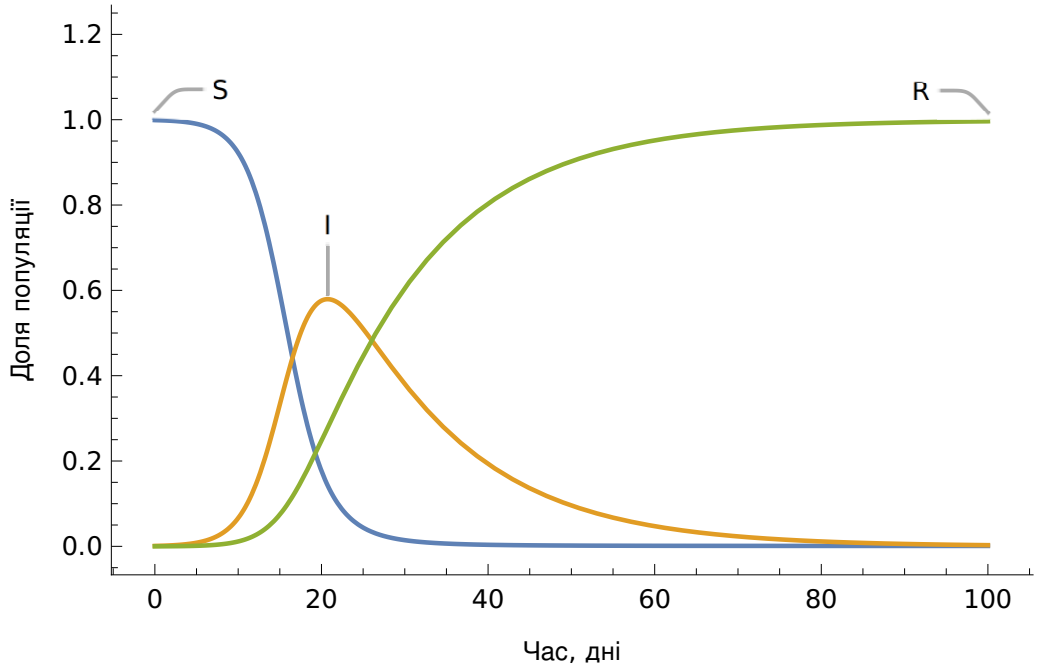

Рис. 1.1: Динаміка розповсюдження інфекції за моделлю SIR

Серед сучасних досліджень, модель SIR була використана для моделювання

розповсюдження COVID-19 та аналізу ефективності карантинних заходів щодо нього [20]. Автори роботи використали модифіковану версію моделі SIR, що передбачала наявність серед представників групи *R* тих, що можуть інфікувати групу *S*.

Серед недоліків моделі SIR є її неможливість моделювати відсутність імунітету до захворювання. Для урахування таких типів захворювань може бути застосована модель SIS. Ця модель відрізняється від SIR тим, що члени групи I після одужання повертаються до групи S. Залежність представників цих груп може бути представлена системою рівнянь:

$$\begin{cases} S'(t) = -\beta S(t)I(t) + \gamma I(t) \\ I'(t) = \beta S(t)I(t) - \gamma I(t) \end{cases}, \tag{1.2}$$

де параметри γ та β аналогічні моделі SIR.

На рисунку 1.2 зображені графіки розв'язання задачі Коші цієї системи при: $\beta = 0.5, \gamma = \frac{1}{14}$ з початковими умовами: $S(0) = 0.999$, $I(0) = 0.001$.

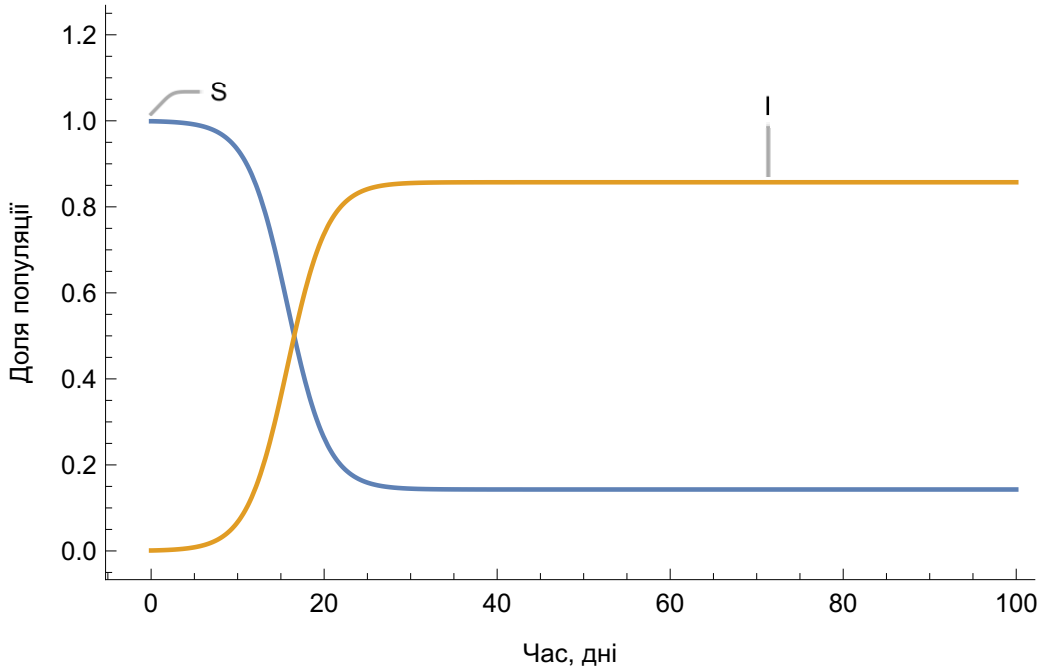

Рис. 1.2: Динаміка розповсюдження інфекції за моделлю SIS

Автори [21] запропонували решіткову модель SIS для дослідження впливу

просторової конфігурації регіонів догляду за хворими на динаміку розповсюдження інфекційного захворювання. У роботі [22] модель SIS була використана для моделювання розповсюдження інфекційних захворювань авіарейсами.

Об'єднанням моделей SIR та SIS є модель SIRS. Ця модель передбачає наявність тимчасового імунітету до захворювання. У рівнянні 1.3 наведене формулювання цієї моделі.

$$\begin{cases} S'(t) = -\beta S(t)I(t) + \alpha R(t) \\ I'(t) = \beta S(t)I(t) - \gamma I(t) \\ R'(t) = \gamma I(t) - \alpha R(t) \end{cases}, \tag{1.3}$$

де α – темп втрати імунітету, що показує щоденну долю членів групи R, що втрачають імунітет; параметри γ та β аналогічні моделі SIR.

На рисунку 1.3 зображені графіки розв'язання задачі Коші цієї системи при: $\alpha = \frac{1}{30}, \beta = 0.5, \gamma = \frac{1}{14}$ з початковими умовами: $S(0) = 0.999$, $I(0) = 0.001$, $R(0) = 0$.

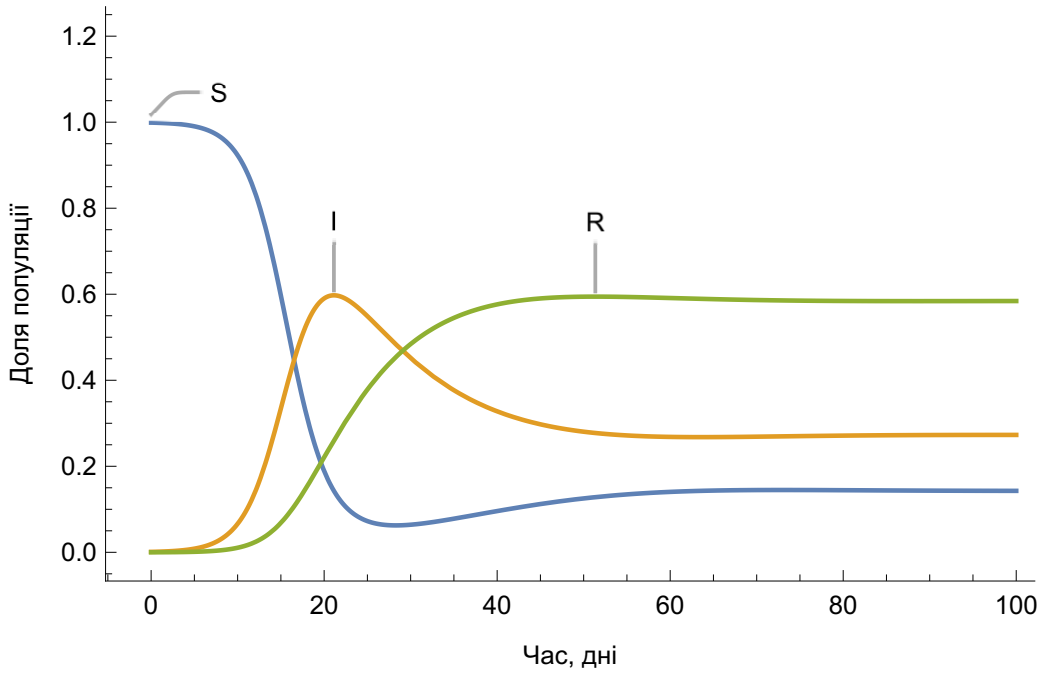

Рис. 1.3: Динаміка розповсюдження інфекції за моделлю SIRS

Варто зазначити, що модель SIRS є узагальненням моделей SIR та SIS через

те, що є еквівалентною першій за умови $alpha = 0$ та другій при $\alpha \to \infty$.

Авторами [23] модель SIRS була використана для моделювання розповсюдження гарячки Денге у Південному Сулавесі. Запропонована модель відрізняється окремими групами популяції, до яких відносяться комахи, що є переносниками захворювання. Модель SIRS також була використана щодо процесу розповсюдження COVID-19 [24].

Для деяких захворювань характерний так званий латентний період, що наступає зразу після інфікування. Він характеризується тим, що хворий попри факт інфікування не проявляє симптомів та не може інфікувати інших. Для побудови моделей таких захворювань до моделей додається група *E* - Exposed. Так, при модифікації моделі SIRS буде отримана модель SEIRS, описана системою рівнянь 1.4.

$$\begin{cases} S'(t) = -\beta S(t)I(t) + \alpha R(t) \\ E'(t) = \beta S(t)I(t) - \sigma E(t) \\ I'(t) = \sigma E(t) - \gamma I(t) \\ R'(t) = \gamma I(t) - \alpha R(t) \end{cases}, \tag{1.4}$$

де σ – темп появи симптомів, що показує щоденну долю членів групи *E*, у яких з'являються симптоми; параметри α, β та γ аналогічні моделі SIRS. На рисунку 1.4 зображені графіки розв'язання задачі Коші цієї системи при: $\alpha = \frac{1}{30}, \beta = 1, \gamma = \frac{1}{14}, \sigma = \frac{1}{14}$ з початковими умовами: $S(0) = 0.999$, $I(0) = 0.001$, $E(0) = R(0) = 0$.

Варто зазначити, що аналогічно моделі SIRS, при $\alpha = 0$ моделюється захворювання з постійним імунітетом (модель SEIR), при $\alpha \to \infty$ – з відсутністю імунітету (модель SEIS).

Просторова модель SEIR на основі графів була використана для моделювання розповсюдження коклюшу [25] у штаті Небраска авторами [26]. Дослідники виконали порівняння точності моделювання розповсюдження цього захворюва-

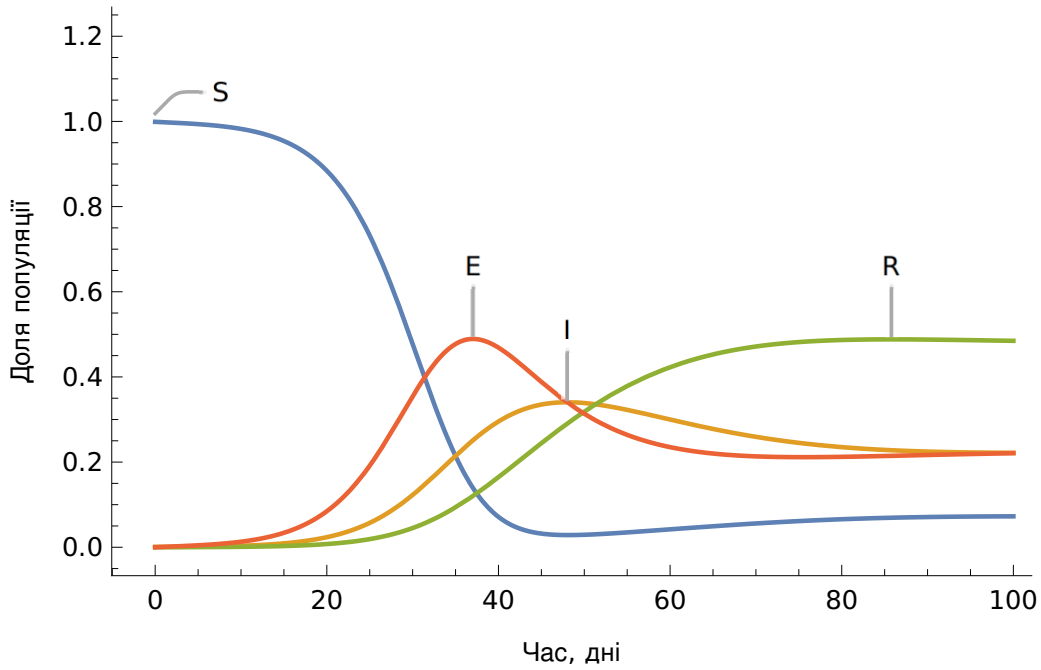

Рис. 1.4: Динаміка розповсюдження інфекції за моделлю SEIRS

ння класичною моделлю SEIR та модифікованою і помітили значне завищення кількості хворих, притаманне класичній моделі, але відсутнє у разі використання модифікованої.

Модель SEIR була використана для моделювання розповсюдження COVID-19 у багатьох країнах та регіонах світу [27, 28, 29, 30, 31].

Для моделювання певних захворювань може бути використана спеціально адаптована під це захворювання модель. Так, автори [32] адаптували модель SEIR під механізм розповсюдження вірусу Зіка [33]. Розроблена модель окрім груп населення ураховувала стани життєвого циклу комах, що є розповсюджувачами вірусу. Розширена модель SEIR була використана для моделювання розповсюдження ВІЛ [34] серед гомосексуальних чоловіків, що є значною групою носіїв цього вірусу серед населення КНР [35].

Одними з часто використовуваних модифікацій моделі SEIR, є моделі, що ураховуються процес вакцинації. Так, подібна модель була використана для оцінки впливу вакцинування на розповсюдження COVID-19 авторами [36], грипу – авторами [37].

Компартментні моделі розповсюдження епідемій можуть ураховувати природну народжуваність та смертність. Для цього, зазвичай додається параметр μ природного приросту населення. Так, модель SEIRS з урахуванням народжуваності та смертності описується системою рівнянь 1.5.

$$\begin{cases} S'(t) = \mu - \mu S - \beta S(t)I(t) + \alpha R(t) \\ E'(t) = \beta S(t)I(t) - (\mu + \sigma)E(t) \\ I'(t) = \sigma E(t) - (\mu + \gamma)I(t) \\ R'(t) = \gamma I(t) - (\mu + \alpha)R(t) \end{cases}, \tag{1.5}$$

де параметри α, β, γ та σ аналогічні звичайній моделі SEIRS. На рисунку 1.5 зображені графіки розв'язання задачі Коші цієї системи при: $\alpha = \frac{1}{30}, \beta = 1, \gamma = \frac{1}{14}, \sigma = \frac{1}{14}, \mu = 0.05$ з початковими умовами: $S(0) = 0.999$, $I(0) = 0.001$, $E(0) = R(0) = 0$.

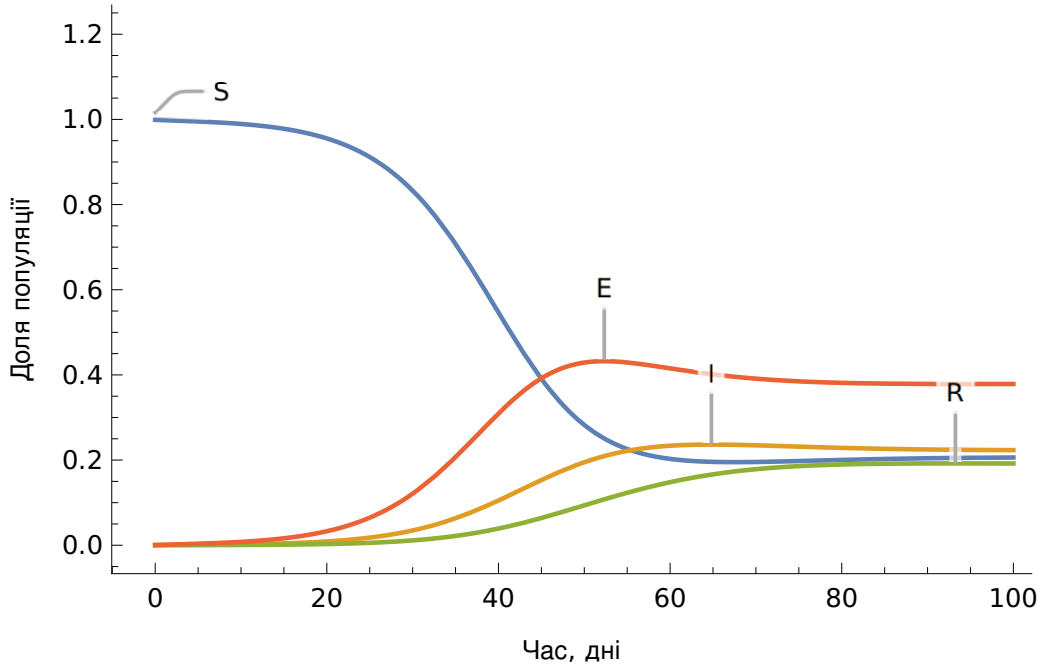

Рис. 1.5: Динаміка розповсюдження інфекції за моделлю SEIRS з урахуванням природного приросту населення

Для моделювання захворювань, носії яких можуть інфікувати інших представників популяції, не маючи при цьому видимих симптомів, група I розбива-

ється на підгрупи I_s та I_a. Представники першої проявляють симптоми захворювання, другої – ні. У разі такої модифікації моделі SEIRS, отримана модель - SEI_aI_sRS описується системою рівнянь 1.6

$$\begin{cases} S'(t) = -S(t)(\beta_a I_a(t) + \beta_s I_s(t)) + \alpha R(t) \\ E'(t) = S(t)(\beta_a I_a(t) + \beta_s I_s(t)) - \sigma E(t) \\ I'_a(t) = (1-p)\sigma E(t) - \gamma I_a(t) \\ I'_s(t) = p\sigma E(t) - \gamma I_s(t) \\ R'(t) = \gamma(I_a(t) + I_s(t)) - \alpha R(t) \end{cases}, \tag{1.6}$$

де параметри α, γ та σ аналогічні звичайній моделі SEIRS, p - ймовірність прояву симптомів у разі інфікування, β_s, β_a – темпи розповсюдження захворювання для симптоматичної та асимптоматичної групи відповідно. На рисунку 1.6 зображені графіки розв'язання задачі Коші цієї системи при: $\alpha = 0, \beta_s = 1, \beta_a = 0.5, \gamma = \frac{1}{14}, \sigma = \frac{1}{14}, p = 0.2$ з початковими умовами: $S(0) = 0.999$, $I_s(0) = 0.001$, $E(0) = I_a(0) = R(0) = 0$.

Авторами [38] була використана SEI_aI_sRS для моделювання розповсюдження COVID-19. За допомогою розробленої моделі було обчислене базове репродукційне число (R_0)[39], та, отримавши величину рівну 2.3, автори зробили висновок, що без заходів протидії пандемія не зникне.

1.2 Агентні моделі

Агентне моделювання – підхід до моделювання, побудований на симуляції поведінки та взаємодії окремих представників модельованого процесу – агентів, з ціллю відобразити динаміку системи в цілому та вплив окремих частин на неї. Агенти, що можуть представляти як окремі фізичні сутності (наприклад люди), так і їх угрупування (наприклад спільноти, організації), мають свій стан, представлений певним набором параметрів, значення яких можуть еволюціонувати

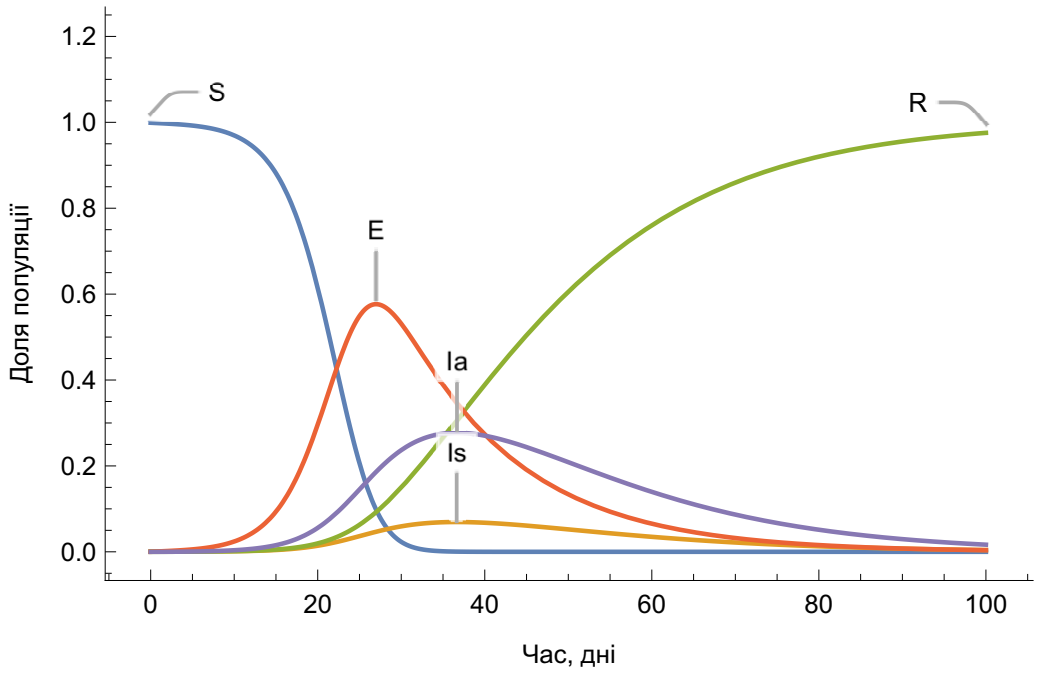

Рис. 1.6: Динаміка розповсюдження інфекції за моделлю SEI_aI_sRS

з плином часу та як результат агентної взаємодії. Такий рівень гранулярності робить можливим як моделювання складних стохастичних процесів, так і аналіз впливу локальних факторів та подій на глобальну динаміку процесу.

Моделі, отримані у процесі застосування агентного моделювання мають назву агентні моделі (agent-based models, ABMs).

Протягом останніх років серед наукових робіт спостерігається збільшення долі присвячених агентним моделям у задачі моделювання розповсюдження інфекційних захворювань. Цей зріст може обґрунтований збільшенням обсягу доступних обчислювальних ресурсів, адже використання агентних моделей як загалом, так і для розв'язання епідеміологічних задач, перш за все лімітоване саме ним. Так, у разі використання тривіального підходу до побудови алгоритму симуляції, обрахунок попарних контактів агентів потребує $N^2/2$ кроків, де N – кількість агентів, що може сягати значних величин у разі побудови глобальних моделей. Ця величина кроків на один крок симуляції може бути надалі збільшена у випадку збільшення часової дискретності моделі.

З проблеми обмеженості агентних моделей доступними обчислювальними

ресурсами, випливає ще одна, не менш важлива проблема – високий рівень необхідних для реалізації цих моделей навичок програмування. Адже у разі реалізації цих моделей недосвіченим розробником слід очікувати значного збільшення часу обрахунку моделі, що, як можна побачити далі, і без цього може сягати значних величин для випадку глобальних моделей.

Попри зазначені проблеми, переваги агентних моделей над компартментними є очевидними. Симуляція кожного представника популяції окремо, на відміну від самої популяції в цілому відкриває можливості урахування як складних стохастичних процесів, так і створення моделей з неоднорідною будовою, що є характерною ознакою реального світу. Про важливість цих переваг свідчить факт того, що агентні моделі здатні до більш точного моделювання процесу розповсюдження інфекційних захворювань ніж компартментні [40, 41].

На відміну від моделювання груп симульованої популяції через аналіз динаміки змін долі представників цих груп від загального їх числа, що характерне компартментним моделям, агентні моделі ураховують належність агентів кожній з групи через присвоєння параметра стану захворювання. Цей стан може показувати чи був агент інфікований, на якому етапі протікання хвороби він знаходиться тощо.

Як і компартментні моделі, моделі агентного типу були використані дослідниками для моделювання широкого спектра сценаріїв розповсюдження інфекційних захворювань, таких як: віспи у разі біотерористичної атаки [42], туберкульозу [43], пташиного грипу (H5N1) [44], звичайного грипу [45], корі [46], малярії [47], COVID-19 [48, 49, 50].

Авторами [51] було виконане дослідження щодо використання різних інструментів для побудови агентних моделей, у тому числі й епідеміологічних. Серед розглянутих інструментів були Repast [52], Swarm [53] та, зокрема для побудови епідеміологічної моделі, NetLogo [54] та СУБД Oracle Database [55]. Досліджена епідеміологічна модель розглядає популяцію агентів, що об’єднані

у контактні групи. Кожен агент може бути присутнім у декількох контактних групах, залишати їх з плином часу та може бути доданий у нові. Кожен крок симуляції відбувається обчислення контактів у межах контактних груп, та відповідно, передача захворювання від інфікованих агентів у разі наявності таких. Модель захворювання представлена трьома станами хвороби, що відповідаю компартментам моделі SIR. Ця тестова модель була реалізована за допомогою NetLogo та Oracle Database та було виконане порівняння швидкості її роботи для кожного з інструментів. Надалі ці реалізації будуть названі AvilovNetLogo та AvilovOracle відповідно. У результаті, був зроблений висновок що попри низький рівень навичок програмування, необхідний для опанування AvilovNetLogo, швидкість її роботи унеможливлює використання для побудови моделей із значною кількістю агентів. AvilovOracle є повною протилежністю, поєднуючи у собі високий поріг входу та високу швидкість роботи (автори доповідають про пришвидшення обчислення кроку моделі у 76-1906 разів в залежності від кількості контактних груп і агентів).

Автори [56] запропонували агентну модель для опису розповсюдження пандемії грипу у межах територій 37 країн Європейського Союзу. Модель розглядає суспільство як множину окремих індивідуумів численністю 515 мільйонів осіб, кожен з яких має набір параметрів, що описують його вік, зайнятість, і т.п. Ці параметри є складовою соціодемографічної параметризації моделі, впливу якої на процес моделювання автори приділили це дослідження. Окрім агентів, у моделі описані географічні місця перебування агенті під час кожного дня симуляції, які представлені місцями роботи, закладами освіти, поділеними на рівні, та житловими будинками. Кожне з цих місць має кількість агентів, що відповідає статистичним даним відповідної країни, наданими Eurostat [57]. Внутрішня мобільність агентів моделюється шляхом відвідувань ними закріплених за ними місць проживання, роботи або навчання. Зовнішня мобільність урахована у моделі шляхом долучення агентів до авіарейсів та подорожей залізною дорогою

до інших країн Європейського Союзу. Окрім цього, модель ураховує загальний транспортний потік пасажирів з або до країн поза межами ЄС. Статистичні розподіли щодо параметрів цих подорожей побудовані на основі відповідних статистичних даних реального світу. Запропонована модель інфекції є дискретним варіантом моделі SEI_aI_sR. Автори доповідають високий рівень швидкодії програмної реалізації розробленої моделі, що дозволяє проводити повну симуляцію за 40 хвилин з витратами оперативної пам'яті у розмірі 9 Гб на комп'ютері з одним процесором Intel Xeon 2.4-GHz. На жаль, не було виконане уточнення щодо кількості кроків цієї симуляції, проте, з представлених графічних додатків можна зробити висновок що кількість кроків лежала у межах від 50 до 200.

F^2-QMRA - модель агентного типу, розроблена для моделювання існування та розповсюдження мікроорганізмів у межах закладів харчування [58]. Кожен заклад освіти представлений у вигляді графу з зонами у вершинах (зона обробки їжі, офіси тощо) та ребрами, що зображають шляхи переміщень між ними. За кожною зоною закріплюються окремі кімнати, що мають певний набір параметрів, таких як площа поверхні, доля поверхонь, що мають пори або тріщини, кількість поверхонь, що можуть брати учать у перехресному зараженні тощо. Ця структура може бути налаштована користувачем для адаптації під конкретний заклад.

Агенти, що діють у межах модельованого закладу мають погодинний розклад відвідувань зон цього закладу, у межах якого контактують з поверхнями, сприяючи забрудненню.

Автори F^2-QMRA приділили особливу увагу процесу життєдіяльності мікроорганізмів. Так, модель ураховує окремо кількість мікроорганізмів, що перебувають зовні на поверхнях та усередині пор та тріщин цих поверхонь. Таке виокремлення різних зон їх перебування дозволяє симулювати процес знезараження більш точно, адже логічним є припущення, що процес прибирання більш ефективний для зовнішньої частини поверхонь. Автори моделюють процес по-

ширення мікроорганізмів поверхнями як внаслідок їх виходу із тріщин, так і шляхом розмноження, темпи якого залежать від температури, pH-рівня та вологості навколишнього середовища.

І хоча F^2-QMRA не є моделлю розповсюдження інфекційних захворювань у строгому сенсі, вона може бути розглянута у рамках цієї роботи через те, що демонструє методи моделювання розповсюдження мікроорганізмів, що можуть буду збудниками інфекційних хвороб у тому числі COVID-19 [59].

1.3 Предметно-орієнтовані мови програмування

Основним засобом реалізації моделей, у тому числі процесу розповсюдження інфекційних захворювань у межах комп'ютерних систем є мови програмування. Мовою програмування є штучна формальна мова, що використовується для задання комп'ютерних програм [60]. Будучи формальною мовою, мова програмування характеризується множиною слів над алфавітом вхідних символів та правил формального виводу, що об'єднані у її граматику [61]. Процес перетворення програми, реалізованою певною мовою програмування у програму іншою мовою (у тому числі мовою машинних команд цільової платформи) називається трансляцією, а програма, що виконує цей процес – транслятором [60].

Історія мов програмування розпочинається з кінця першої половини XX сторіччя. Так, однією з перших мов програмування, що відійшла від концепції використання машинних команд можна назвати мову Short Code [62]. Ця мова дозволяла задавати математичні вирази, обрахування яких транслювалось у машинний код для подальшого виконання. За всю історію існування людства була розроблена значна кількість мов програмування. І хоча точна величина є невідомою, перерахунок статей з онлайн-енциклопедії Вікіпедія дає оцінку в 700 одиниць, у той час, як онлайн-енциклопедія мов програмування HOPL налічує 8945 одиниць [63, 64].

Мови програмування можуть бути класифіковані за багатьма ознаками. Так,

основним є розбиття за парадигмою програмування, тобто за способом концептуалізації, що визначає як треба проводити обчислення та як робота, що виконує комп'ютер, має бути структурована та організована [65]. Суміжним є розбиття на мови програмування загального призначення та предметно-орієнтовані. До першої категорії відносять ті мови, що були спроєктовані для вирішення широкого спектра задач у рамках багатьох предметних областей. Прикладами таких мов є мови C/C++, Java, Python. Предметно-орієнтованим мовам характерне проєктування з метою їх використання у конкретній предметній області для вирішення вузького діапазону задач, нерідко навіть однієї конкретної задачі. Прикладами є мова опису шейдерів GLSL [66], мова системи автоматичного збирання програмного забезпечення Gradle [67], мова компартментного інфекційного моделювання Kendrick [68]. Спеціалізованість цих мов для використання у певній предметній області може бути виражене в абстракціях дуже високого рівня, унікальних для цієї мови, відсутності інструкцій, сутностей, характерних іншим мовам програмування. Так, мова Kendrick дозволяє задання сутності "інфекційне захворювання", що, очевидно, не є доцільним використовувати поза межами епідеміології та ряду суміжних дисциплін. Використання таких високоабстрактних сутностей та позбавлення мови низькорівневих інструкцій дозволяє значно спростити програмування цією мовою та зменшити час розробки програм.

1.4 Програмні засоби моделювання

Серед програмних засобів моделювання розповсюдження інфекційних захворювань були виділені дві категорії: програмні засоби побудови компартментних моделей та агентних моделей.

Серед представників першої категорії можна виділити програмний компонент та однойменну мову моделювання Kendrick [68]. Мова Kendrick надає можливість описувати моделі компартментного типу через непряме задання від-

повідних систем диференційних рівнянь та початкових умов. Так, у лістингу нижче наведений приклад опису SEIR-моделі для опису розповсюдження кору:

```
KendrickModel SEIR
attribute: #( status -> S E I R );
parameters: #( beta lambda gamma sigma mu );
transitions: #(
S -- lambda --> E.
E -- sigma --> I.
I -- gamma --> R.
status -- mu --> Empty.
Empty -- mu --> S.
) .

Composition Measles
model: 'SEIR'.

Scenario MeaslesParameters
on: 'Measles';
beta: 0.0000214;
gamma: 0.143;
mu: 0.0000351;
sigma: 0.125;
lambda: #( beta*I ).

Scenario MeaslesPopulation
on: 'Measles';
populationSize: 100000;
S: 99999;
I: 1;
others: 0.
```

Програмний компонент Kendrick являє собою середовище програмування, що дозволяє виконувати програмування та аналіз побудованих моделей. Зовнішній вигляд інтерфейсу користувача представлений на рисунку 1.7.

Користувацький інтерфейс середовища програмування має просту трикомпонентну структуру: ліва частина надає призначення для роботи зі файловою системою користувача, центральна – для перегляду та редагування програмно-

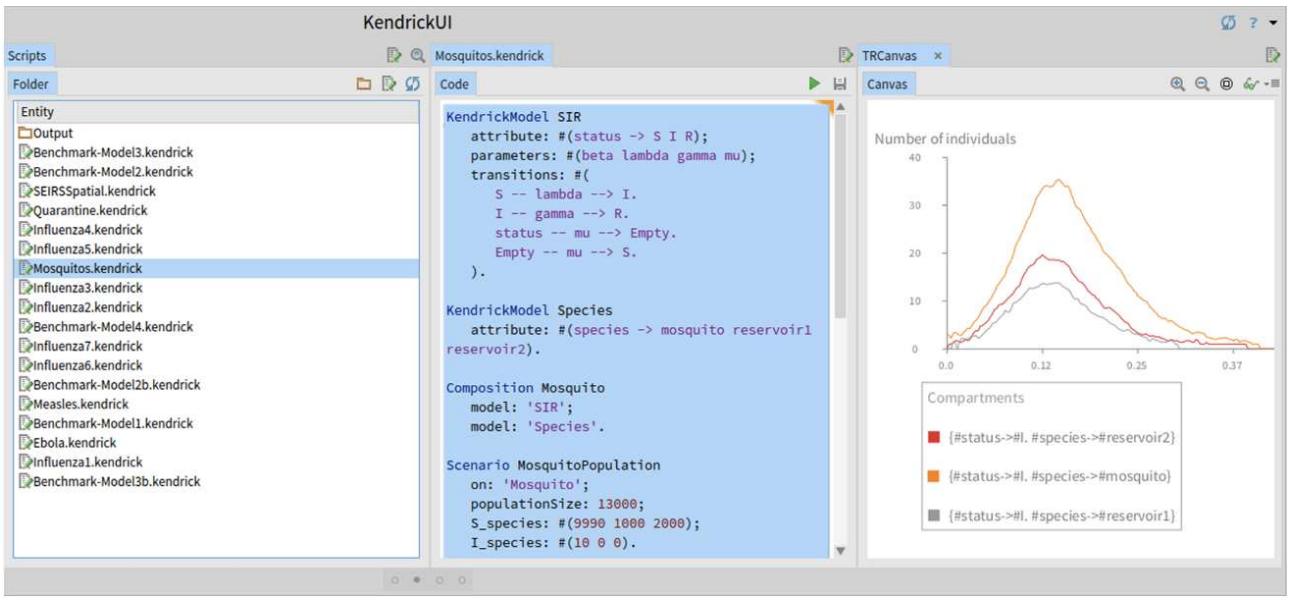

Рис. 1.7: Користувацький інтерфейс середовища програмування Kendrick

го коду, права – для аналізу побудованої моделі.

До програмних засобів побудови компартментних епідеміологічних моделей можна також віднести системи комп'ютерної алгебри, такі як: Wolfram Mathematica[69], Maxima[70]. Так, SEIR-модель, подібна до представленої у лістингу мовою Ke-ndrick, реалізована у середовищі Wolfram Mathematica представлена на рисунку 1.8.

І хоча системи комп'ютерної алгебри надають широкі можливості побудови математичних моделей, необхідні навички програмування лімітують можливу аудиторію їх користувачів. Окрім цього, побудова агентних моделей у рамках цих систем не є практичною.

Мови математичного моделювання такі як Matlab[71], Modelica[72] або Scilab[73] можуть бути використані для побудови епідеміологічних моделей, але їхні широкі можливості призводять до неефективності використання обчислювальних ресурсів щодо даної предметної області.

Окрім мов та середовищ загального моделювання існують також спеціалізовані виключно для побудови епідеміологічних моделей. Так, FluTE [74] - реалізований мовою програмування C/C++ програмний компонент, що побудо-

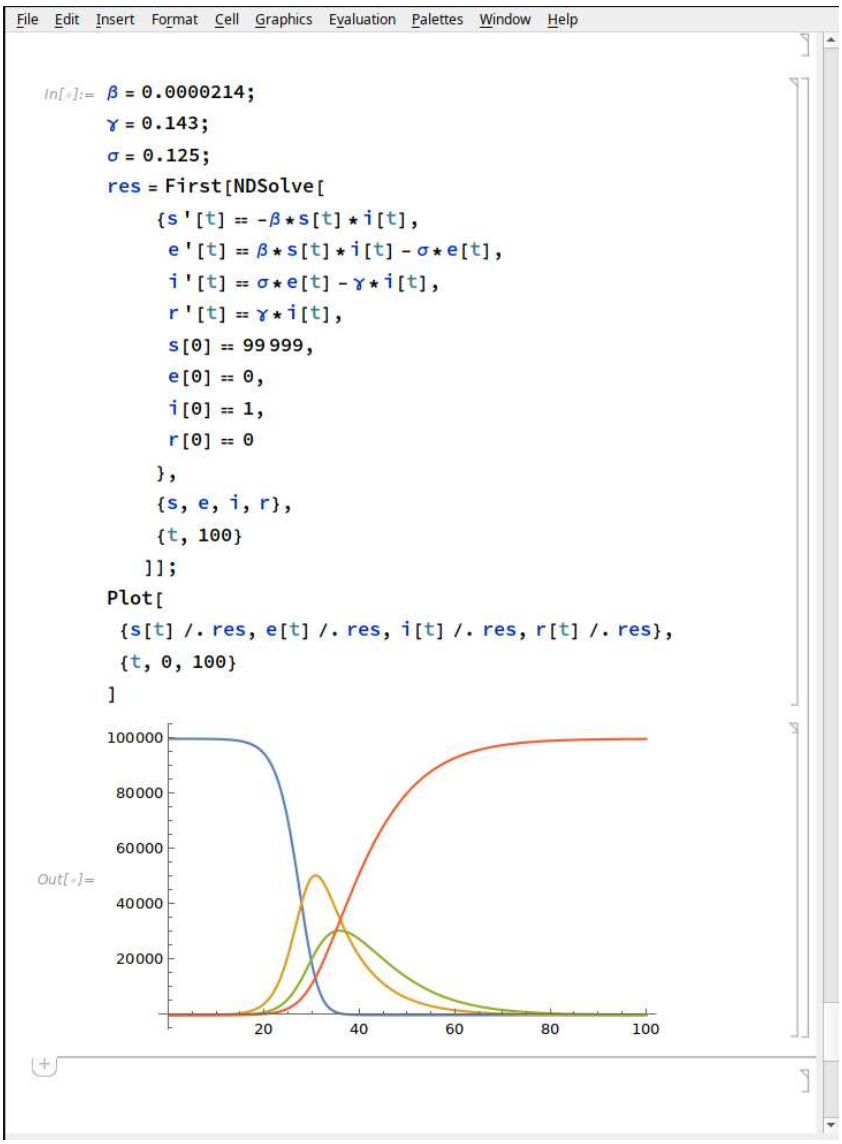

Рис. 1.8: Реалізація SEIR-моделі у середовищі системи комп'ютерної алгебри Wolfram Mathematica

ваний на основі агентної моделі розповсюдження інфекційного захворювання. FluTE моделює типове американське суспільство, поділене на громади по 500-3000 осіб. Кожна громада представлена родинами з 1-7 особами у кожній, що мешкають у приватних будинках. У ночі родичі контактують один з одним, а також з іншими представниками своєї громади, сприяючи цим розповсюдженню захворювання. У день, діти відвідують заклади освіти, що закріплені за кожною громадою та контактують з іншими дітьми у їхніх навчальних групах. Діти дошкільного віку контактують з іншими дітьми, граючи разом з ними на

ігрових майданчиках. У цей час, зайняті дорослі відвідують місця роботи, що закріплені за їхньою або сусідніми громадами, контактуючи там зі своїми колегами. Як дорослі, так і діти можуть долучатися до короткочасних подорожей, що моделюють відпустки та інші подорожі. Процес введення захворювання у суспільство відбувається через зараження випадкової групи населення на початку симуляції, або кожного дня. Усі дані, від вікового розподілу осіб, до розподілу кількості ночей, що особа перебуває у відпустці, побудовані на основі статистичних даних суспільства Сполучених Штатів Америки.

Інфікування відбувається шляхом трасування контактів, з ймовірністю передачі захворювання від однієї особи до іншої каліброваної на даних частоти захворювання для підтипів вірусу H1N1 та H2N2.

Програмному компоненту відповідає конфігураційний файл, що дозволяє параметризувати модель.

Автори використали FluTE для побудови моделі розповсюдження грипу у межах міста Сієтл та всієї території США. На рисунку 1.9 представлені результати моделювання відповідно до FluTE.

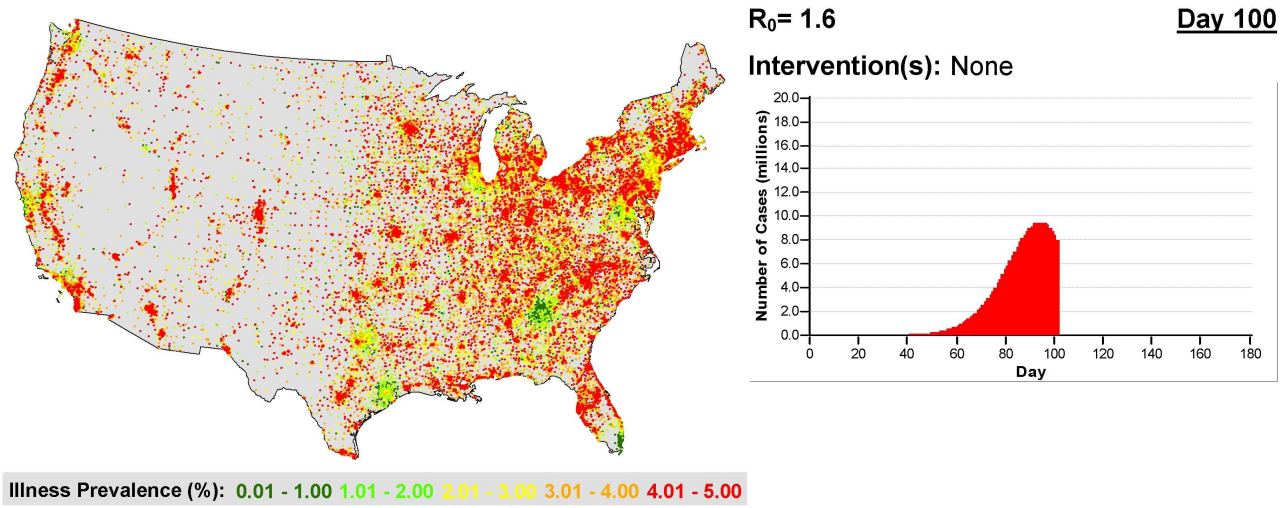

Рис. 1.9: Розповсюдження вірусу у межах території США відповідно до FluTE. Кольором позначена розповсюдженість захворювання

Автори не представили детальну статистику щодо швидкості роботи моделі,

окрім того, що модель з 10 мільйонами осіб може обраховуватись протягом двох годин на процесорі Intel Core2 Duo T9400 (залежно від R_0 інфекції). Також, було заявлено, що моделювання популяції Сполучених Штатів Америки розміром 280 мільйонів осіб займає 192 години. З рисунка 1.9 можна припустити, що моделювання велось протягом 180 днів, що відповідає 1.07 годинам на один крок симуляції.

Global Epidemic and Mobility (GLEaM) та відповідне середовище аналітики - GLEaMvis[75] являє собою програмний комплекс побудований на основі клієнт-серверної архітектури для створення, налаштування та аналітики компартментних моделей. Він складається з трьох основних компонентів: користувацького програмного компонента, проміжного сервісу та движку симуляції. Перший з них надає користувачеві можливість детально налаштовувати компартменти, зв'язки між ними, деталі симуляції та переглядати аналітику щодо результатів роботи моделі. На рисунку 1.10 зображений користувацький інтерфейс інструменту побудови компартментної моделі, на рисунку 1.11 – кроків налаштування симуляції. Проміжний сервіс та движок симуляції відповідні за взаємодію з клієнтським програмним компонентом та роботи епідеміологічної моделі. Компартментна модель, що лежить в основі GLEaMvis складається з трьох шарів: населення, мобільності та епідеміології. Перший шар являє собою статистичні дані щодо населення у вигляді решітки з кроком сітки в 15 кутових мінут. Шар мобільності відповідний за моделювання переміщень населення як на короткі, так і довгі відстані. Епідеміологічний шар виконує моделювання динаміки розповсюдження інфекційного захворювання у межах окремих громад відповідно до заданої користувачем компартментної моделі.

GLEaMviz надає користувачу можливість переглядати детальну статистику щодо роботи моделі у наочній візуалізації, приклад якої наведений на рисунку 1.12.

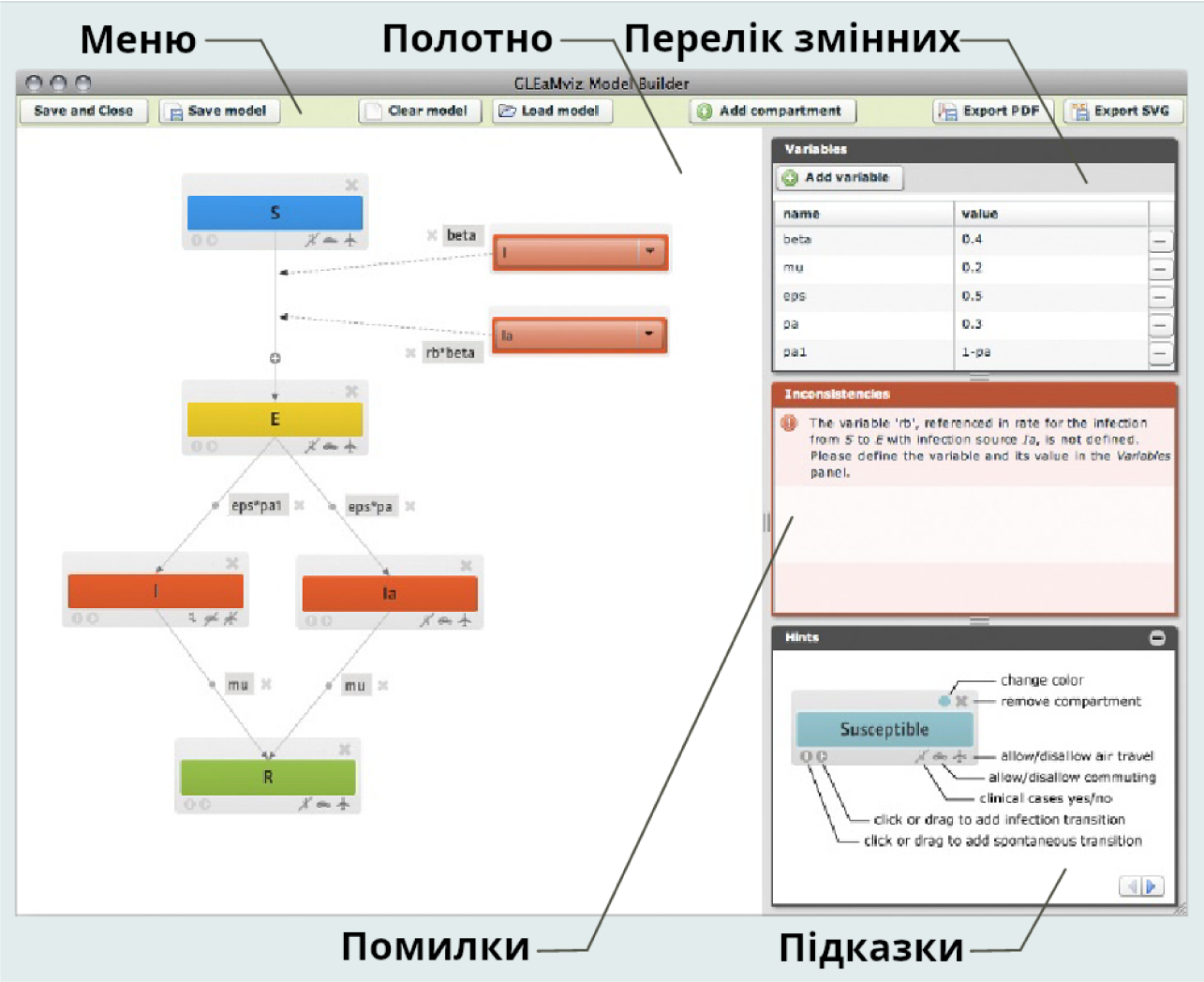

Рис. 1.10: Користувацький інтерфейс інструменту побудови компартментної моделі шляхом задання компартментів, зв’язків між ними та параметрів у середовищі моделювання GLEaMvis

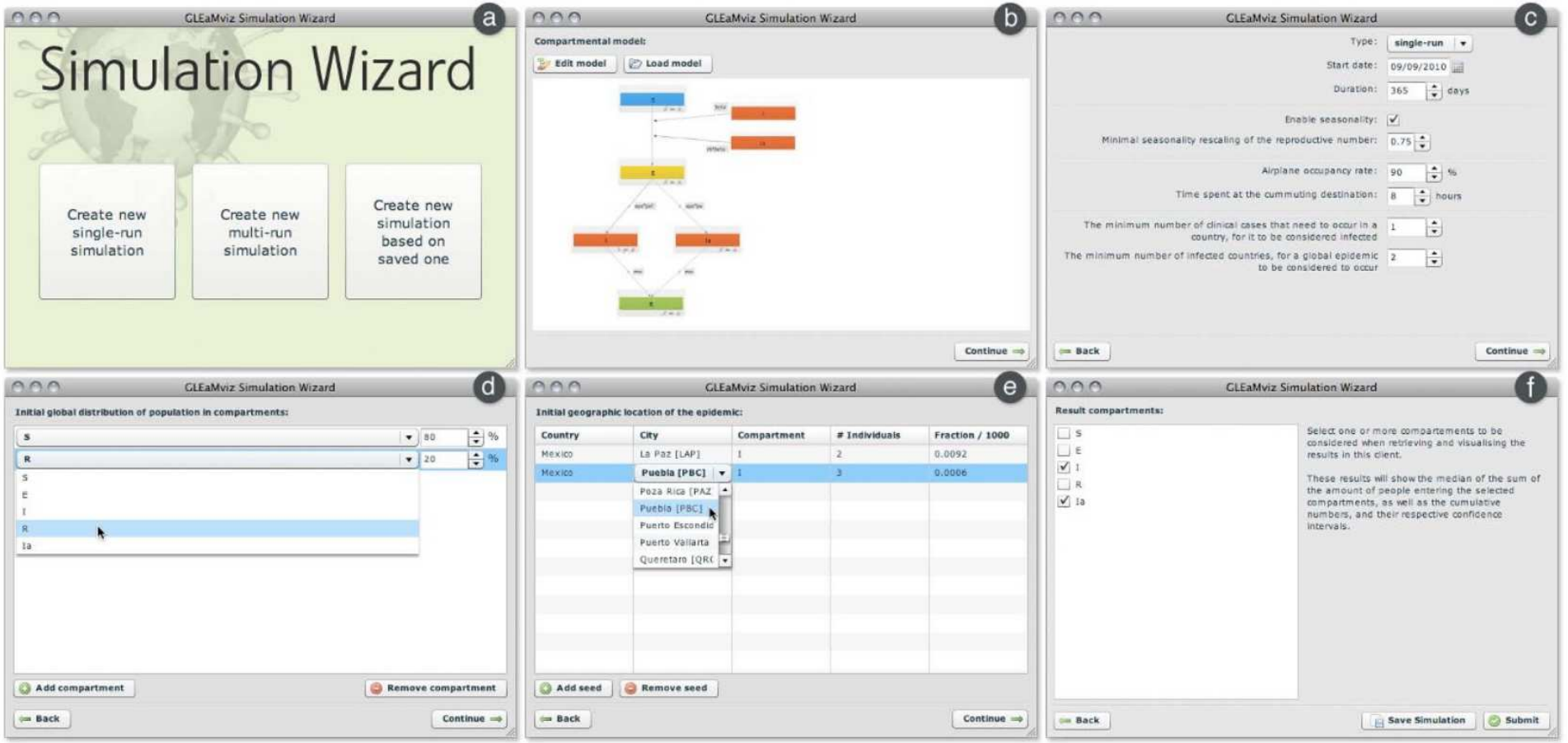

Рис. 1.11: Користувацький інтерфейс інструменту налаштування симуляції у середовищі GLEaMvis. **a** - вікно вибору типу симуляції; **b** - вікно вибору компартментної моделі; **c** - вікно налаштування параметрів симуляції; **d** - вікно налаштування початкового розміру компартментів; **e** - вікно налаштування географії епідемії; **f** - вікно вибору компартментів для представлення в аналітиці

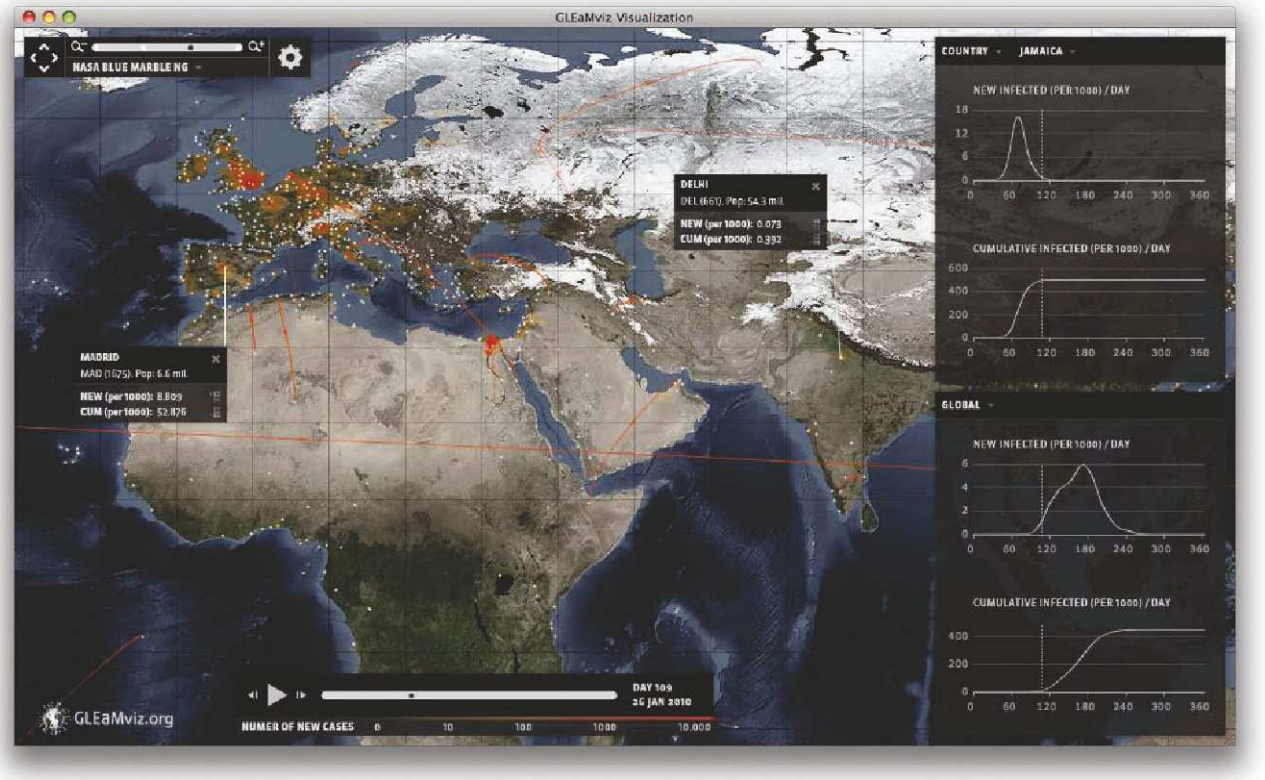

Рис. 1.12: Користувацький інтерфейс інструменту аналітики у середовищі моделювання GLEaMvis

Spatiotemporal Epidemiological Modeler (STEM) [76] являє собою реалізоване мовою програмування Java середовище моделювання розповсюдження інфекційних захворювань. Епідеміологічна модель STEM побудована на основі графів для представлення географічних локацій та транспортних потоків між ними. За допомогою вбудованого редактора графів (рисунок 1.13 користувач може створювати шар мобільності розробленої моделі. Цей шар може включати декілька графів, кожен з яких може відповідати, наприклад, різним типам транспорту. Вбудований редактор множини компартментів та зв'язків між ними дозволяє розробляти користувацькі компартментні моделі. У цілях аналітики, STEM дозволяє візуалізувати процес розповсюдження інфекційного захворювання на географічній карті території моделювання.

Framework for Reconstructing Epidemic Dynamics (FRED) [77] використовує агентну модель для моделювання розповсюдження інфекційних захворювань.

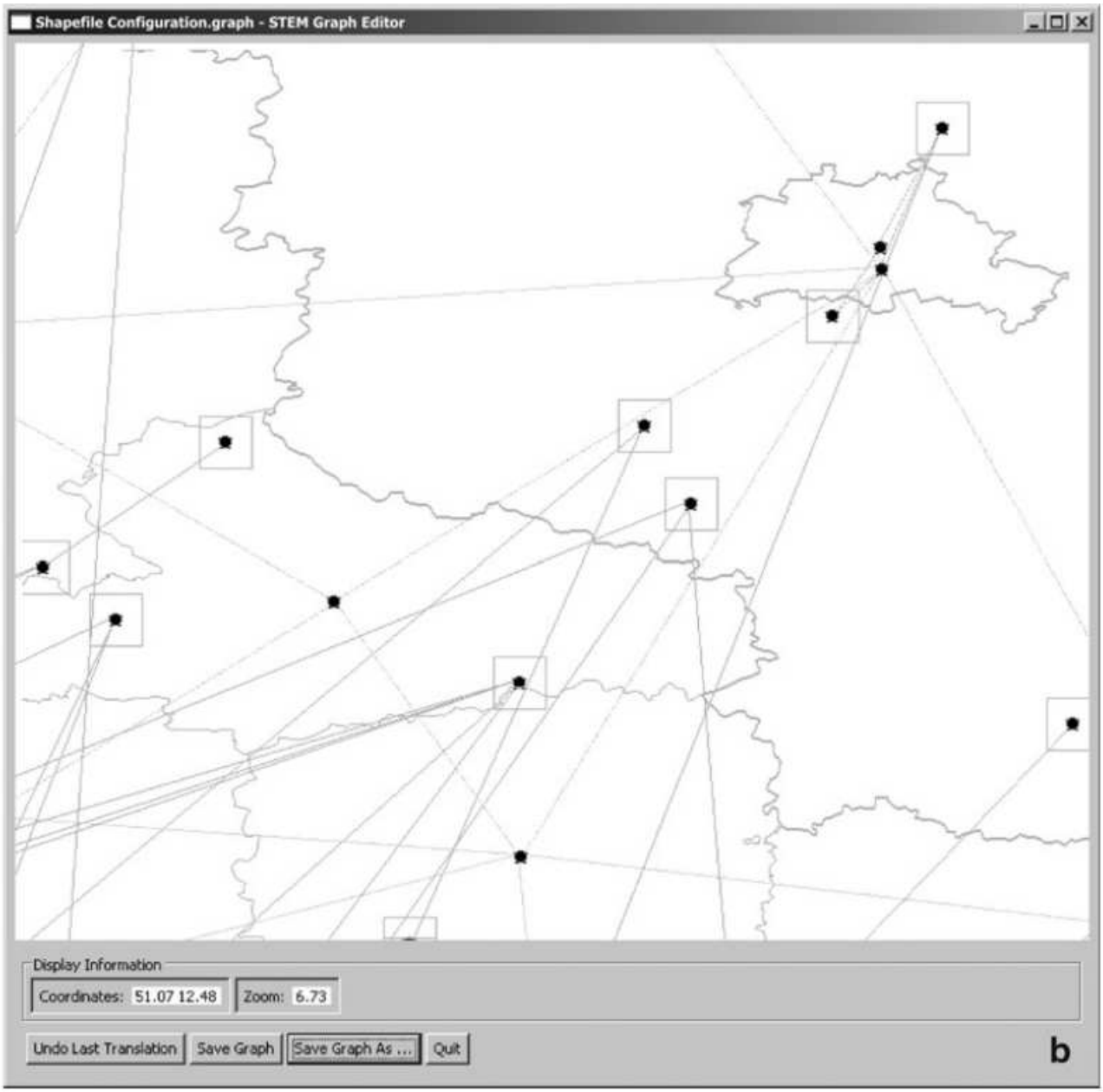

Рис. 1.13: Користувацький інтерфейс інструменту редагування графів у середовищі моделювання STEM

Подібно до FluTE, цьому методу притаманне глибоке моделювання соціодемографічної структури населення з урахуванням таких рис кожного агенту як стать, вік, раса, зайнятість та ін., для урахування як у процесі щоденних переміщень, так і процесу протікання захворювання. У рамках переміщень, агент щоденно відвідує певні установи, у яких контактує з іншими агентами, сприяючи розповсюдженню захворювання. І хоча у користувача немає можливості редагувати наявні та додавати нові типи цих установ, FRED дозволяє налаштовувати темпи інфікування агентів для кожного з них.

Для своєї роботи FRED потребує від 750 до 1000 Мб оперативної пам’яті на один мільйон агентів. Час роботи моделі, вказаний авторами, може варіюватись від двох хвилин на один мільйон агентів для типового двоядерного процесора та 4-х годин для 16-ядерного процесора у разі роботи моделі з 316 мільйонами агентів. На жаль, автори не вказали кількість днів у кожній симуляції, проте, по аналогії з іншими опублікованими їми моделями, можна припустити що це значення лежить у межах від 50 до 200.

У таблиці 1.1 наведено порівняння доступного в розглянутих методах та інструментах епідеміологічного моделювання функціонала.

Таблиця 1.1: Порівняння доступного функціоналу у наявних методах/інструментах епідеміологічного моделювання з середовищем CTraceEnv

	FluTE	FRED	NetLogo	GiacopelliLombardy	Kendrick	GLEaM	STEM
Компартментні моделі	-	-	-	-	+	+	+
Агентні моделі	+	+	+	+	-	-	-
Задання користувацьких станів захворювання/компартментів	-	-	+	-	+	+	-
Урахування атрибутів окремих представників популяції	+	+	+	+	-	-	-
Задання атрибутів представників популяції	+	-	+	-	+	-	-
Динамічні значення атрибутів представників популяції	-	-	+	-	+	+	+
Нефіксована просторова роздільна здатність	-	-	+	-	-	-	+
Нефіксована часова роздільна здатність	-	-	+	-	-	-	-
Вбудована Інтеграція з джерелами статистичних даних	-	-	-	-	-	-	+
Паралельне/розподілене обчислення	+	+	-	+	-	+	+
Графічний інтерфейс користувача	-	+	+	-	+	+	+
Динамічні параметри моделі	-	-	+	-	-	+	-
Нефіксовані типи місць	-	-	+	-	-	-	-
Векторні захворювання	-	-	+	-	+	+	+
Моделювання вакцинації	-	+	+	+	+	+	+
Моделювання сезонних явищ	-	+	+	-	+	+	+
Моделювання географії	+	+	+	+	-	+	+
Урахування погодних умов/клімату	-	-	+	-	+	-	-

ВИСНОВКИ ДО ПЕРШОГО РОЗДІЛУ

У рамках першого розділу була опрацьована література за темою математичних, програмних та мовних засобів моделювання розповсюдження інфекційних захворювань з метою виокремлення основних дослідницьких напрямків та аналізу поточних рішень.

Проаналізовані методи були поділені на три групи: моделі компартментного типу, моделі агентного типу та програмні засоби моделювання. Серед основних робіт щодо досліджуваної області було помічене збільшення долі присвячених агентному моделюванню, що свідчить збільшення інтересу до цього типу моделювання. Були виявлені переваги та недоліки запропонованих дослідницькою спільнотою рішень та, зокрема, обсяг, гнучкість та швидкодія програмних засобів моделювання. Серед програмних засобів моделювання не було помічено представників, що поєднують у собі доступний інтерфейс з високою швидкодією та гнучкістю опису моделей. Зазвичай, вони являли собою бібліотеку певної мови програмування, використання якою потребує навичок програмування, або окремий програмний продукт з фіксованою моделлю, інтегрованою у нього. Єдине рішення, що не має цих недоліків — платформа Kendrick, у рамках якої опис компартментної моделі виконується за допомогою предметно-орієнтованої мови програмування.

Узагальнюючи цю інформацію був зроблений висновок, що розробка предметно-орієнтованої мови програмування для опису агентних моделей та графічного середовища розробки та аналізу нададуть доступний інтерфейс опису моделі у поєднанні зі значною гнучкістю та швидкодією.

РОЗДІЛ 2.

МАТЕМАТИЧНИЙ ОПИС ЗАГАЛЬНОЇ АГЕНТНОЇ МОДЕЛІ РОЗПОВСЮДЖЕННЯ ІНФЕКЦІЙНИХ ЗАХВОРЮВАНЬ ТА ВІДПОВІДНА ЇЙ ФОРМАЛЬНА МОВА ОПИСУ

2.1 Загальна агентна модель розповсюдження інфекційних захворювань

Аналіз сучасних методів розв'язання задачі агентного моделювання розповсюдження інфекційних захворювань показав ряд рис, притаманних більшості розглянутих робіт. Агентним моделям характерний покроковий метод симуляції, що передбачає дискретність часу. Наприклад, часовий проміжок між двома сусідніми станами моделі може дорівнювати одній годині, одному дню тощо, та є постійним протягом часу. Позначимо множину кроків моделі як $\mathbf{T} \subset \mathbb{N}$. Для збільшення часової дискретності моделі, виконаємо подальше розбиття кроків симуляції на часові кроки, представлені множиною $\mathbf{H} \subset \mathbb{N}, |\mathbf{H}| \in \mathbb{N}$.

Модель може бути параметризована множиною величин, які керують тими чи іншими властивостями модельованого середовища. Такими параметрами у разі епідеміологічного моделювання може бути, наприклад, час симуляції, кількість агентів, інтенсивність інфікування зовнішніми чинниками тощо. Позначимо множину значень цих параметрів як $\Omega \subseteq \mathbb{R}^n$, де n — їх кількість.

Серед сутностей, що моделюються найчастіше розглядають дві категорії: агенти на місця. Агенти виступають представниками модельованої популяції, розповсюдженню захворювання серед членів якої й присвячена модель. У деяких роботах з групи агентів виокремлюється категорія векторів, для моделювання векторних захворювань. Агенти можуть мати перелік параметрів, значення яких може бути як постійним у часі (стать), так і змінюватись із часом (вік, стан хвороби). Значення цих параметрів впливає на поведінку агента під час

симуляції.

Для спрощення подальших виразів, введемо наступну нотацію:

$$A' = \{(a_2, a_3, \ldots, a_n) : \exists a_1, (a_1, a_2, \ldots, a_n) \in A\} \tag{2.1}$$

Місця моделюють приміщення або зони, у яких агенти можуть контактувати, сприяючи розповсюдженню захворювання. Ними можуть бути квартири, школи, офіси, транспорт тощо. Позначимо **A**, **P** множини агентів та місць відповідно, а через $\mathcal{A} \subseteq \mathbb{R}^n, \mathcal{P} \subseteq \mathbb{R}^m$ — множини їх параметрів, де *n* — кількість параметрів агента, *m* — кількість параметрів місця. Позначимо π функцію, що для заданого агента чи місця, віддає відповідні йому параметри: $\pi : (\mathbf{A} \cup \mathbf{P}) \times \mathbf{T} \to (\mathcal{A} \cup \mathcal{P})$.

Через те, що параметри агентів та місць не є статичними, модель має містити механізм урахування їх динаміки. Введемо функцію *еволюції параметрів* агента (рівняння 2.2) та місця (рівняння 2.3):

$$\upsilon_a : \Omega \times \mathcal{A} \to \mathcal{A} \tag{2.2}$$

$$\upsilon_p : \Omega \times \mathcal{P} \to \mathcal{P} \tag{2.3}$$

Тоді:

$$\begin{aligned} \forall \mathbf{a} \in \mathbf{A}, \forall t \in \mathbf{T}, \pi(\mathbf{a}, t+1) = \pi(\mathbf{a}, t) + \upsilon_a(\omega, \mathbf{a}) \\ \forall \mathbf{p} \in \mathbf{P}, \forall t \in \mathbf{T}, \pi(\mathbf{p}, t+1) = \pi(\mathbf{p}, t) + \upsilon_p(\omega, \mathbf{p}) \end{aligned} \tag{2.4}$$

Зазвичай, агенти відвідують місця, відповідно до певного розкладу, керованого значеннями внутрішніх параметрів агента. Так, наприклад, розклад агента у разі виявлення симптомів хвороби може змінитись на постійне перебування вдома. Позначимо розклад агента як:

$$\phi : \Omega \times \mathcal{A} \times \mathbf{S} \times \mathbf{T} \times \mathbf{H} \to \mathbf{P} \tag{2.5}$$

Таким чином, у рамках моделі, параметризованої $\omega \in \Omega$, агент з параметрами $a \in \mathcal{A}$ та станом захворювання $s \in \mathbf{S}$ у крок симуляції $t \in \mathbf{T}$ у часовий крок $h \in \mathbf{H}$ перебуває у місці $\phi(\omega, a, s, t, h)$.

На відміну від компартментів, призначенням яких є урахування кількості або долі населення, що мають спільний етап протікання захворювання, агентним моделям характерне окреме урахування стану захворювання для кожного агенту. Ці стани зазвичай відповідають компартментам класичних компартментних моделей. Позначимо через $\mathbf{S}$ множину станів захворювання. Ця множина може складатись зі стандартних трьох елементів у разі використання станів, подібних до компартментів моделі SIR: $\mathbf{S} = \{S, I, R\}$, або ураховувати стани одразу декількох популяцій у разі моделювання векторних захворювань. Наприклад $\mathbf{S} = \{S_H, E_H, I_H, R_H, S_V, E_V, I_V, V_1, V_2, V_3\}$ для реалізації агентного аналога моделі запропонованої у [32]. Тоді, динаміка зміни стану захворювання кожного агенту з часом може бути позначена як:

$$\xi : \mathbf{A} \times \mathbf{T} \to \mathbf{S} \tag{2.6}$$

Динаміка зміни стану захворювання кожного агенту може бути обумовлена контактами цього агенту з іншими представниками модельованої популяції. Так, у рамках SIR-подібної моделі, агент зі станом *S*, що перебував в одному місці разом з агентом зі станом *I* може з ненульовою ймовірністю змінити цей стан на наступному кроці на стан *I*. Таким чином, здатність модельованого захворювання до розповсюдження шляхом зараження одних агентів іншими може бути сформульована *функцією розповсюдження* Φ_t, заданою у рівнянні 2.7.

$$\begin{aligned}
&\Phi_t : \Omega \times \mathbf{S} \times \mathcal{A}^2 \times \mathcal{P} \to [0, 1] \\
&\Phi_t(\omega, s, a, d, p) = P(\xi(a, t+1) = s | \phi(\omega, a, t, h) = \phi(\omega, d, t, h) = \mathbf{p}, \pi(\mathbf{p}, t) = p), \\
&\omega \in \Omega, a, d \in \mathcal{A}, t \in \mathbf{T}, h \in \mathbf{H}, \mathbf{p} \in \mathbf{P}
\end{aligned} \tag{2.7}$$

Для спрощення нотації та подальшого проєктування мови опису моделі, Φ_t може бути розбита на множину *правил розповсюдження* $\mathbf{R}_t$. Кожне правило $r \in \mathbf{R}_t$ диктує ймовірність агента-реципієнта змінити стан захворювання на наступному кроці симуляції при контакті з агентом-донором, що може залежати від глобальних параметрів моделі, параметрів обох агентів, місця їх зустрічі і являє собою кортеж:

$$
\begin{aligned}
& r = ((s_a, s_d, s), f) \\
& f : \Omega \times A^2 \times \mathcal{P} \to [0, 1], \\
& \forall a \in A, \exists s \in \mathbf{S} : (s, a_1, \ldots, a_n) \in \mathcal{A} \\
& f(\omega, s, s_a, s_d, a, d, p) = P(\xi(a, t+1) = s| \\
& \phi(\omega, a, t, h) = \phi(\omega, d, t, h) = p, \\
& \xi(a, t) = s_a, \xi(d, t) = s_d, \pi(\mathbf{p}) = p), \\
& \omega \in \Omega, a, d \in A, t \in \mathbf{T}, h \in \mathbf{H}, \mathbf{p} \in \mathbf{P}
\end{aligned}
\tag{2.8}
$$

Зміна стану захворювання агента може відбуватись і без наявності контактів з іншими агентами, що дозволяє моделювати, наприклад, процес одужання. У такому разі, модельованому захворюванню характерна *функція розвитку* Φ_p, задана у рівнянні 2.9 та відповідна множина *правил розвитку* $\mathbf{R}_p$. Кожне правило $r \in \mathbf{R}_p$ є функцією ймовірності зміни стану захворювання агента на наступному кроці симуляції від нового стану, параметрів агенту та часу, що пройшов с моменту останньої зміни стану цього агента і задане у рівнянні 2.10.

$$
\begin{aligned}
& \Phi_p : \Omega \times \mathbf{S} \times \mathcal{A} \times \mathbb{N} \to [0, 1] \\
& \Phi_p(\omega, s, a, \Delta t) = P(\xi(\mathbf{a}, t+1) = s| \\
& \pi(\mathbf{a}) = a, \\
& \forall i, j \in \mathbb{N} \cap [t, t - \Delta t), \xi(\mathbf{a}, i) = \xi(\mathbf{a}, j) \neq \xi(\mathbf{a}, t - \Delta t))), \\
& \omega \in \Omega, \mathbf{a} \in \mathbf{A}, \Delta t \in \mathbb{N}
\end{aligned}
\tag{2.9}
$$

$$
\begin{aligned}
&r : \Omega \times \mathbf{S}^2 \times \mathcal{A} \times \mathbb{N} \to [0,1] \\
&r(\omega, s, s_p, a, \Delta t) = P(\xi(\mathbf{a}, t+1) = s | \xi(\mathbf{a}, t) = s_p, \pi(\mathbf{a}) = a), \\
&\omega \in \Omega, s, s_p \in \mathbf{S}, \mathbf{a} \in \mathbf{A}, \Delta t \in \mathbb{N}
\end{aligned} \tag{2.10}
$$

Беручи до уваги вищесказане, модель інфекційного захворювання може бути сформульована як:

$$
\Delta = \{\mathbf{S}, \Phi_t, \Phi_p\} \tag{2.11}
$$

Використовуючи описану нотацію, загальний алгоритм симуляції представлений в Алгоритмі 1.

$\mathcal{P}_\Delta(\mathbf{a}, t)$ є розподілом майбутнього стану захворювання агента $\mathbf{a} \in \mathbf{A}$ у крок $t+1, t \in \mathbf{T}$, заданим у рівнянні 2.12.

$$
\begin{aligned}
&\mathcal{P}_\Delta(\mathbf{a}, t) : \forall s \in \mathbf{S}, s \neq \xi(\mathbf{a}, t) \\
&\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s) = 1 - (1 - \Phi_p(\omega, s, \pi(\mathbf{a}, t), \tau(\mathbf{a}, t))) \cdot \\
&\cdot \prod_{\mathbf{d} \in \mathbf{A} \setminus \{\mathbf{a}\}} \prod_{\mathbf{p} \in \mathbf{P}} (1 - \Phi_t(\omega, s, \pi(\mathbf{a}, t), \pi(\mathbf{d}, t), \pi(\mathbf{p}, t)))
\end{aligned} \tag{2.12}
$$

2.1.1 Обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$

Обрахунок значень $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s), \forall s \in \mathbf{S}$ є центральною та складною проблемою агентного епідеміологічного моделювання. Її складність полягає у тому, що у тривіальній реалізації алгоритму її вирішення, необхідно розглянути кожну пару агентів $a, d \in \mathbf{A}$, що робить обчислювальну складність алгоритму рівною $O(n^2)$. Це унеможливлює реалізацію моделей глобального масштабу з кількість агентів більшою за 10^6.

У процесі розв'язання означеної проблеми з обчислювальною складністю розглянемо були запроваджені наступні алгоритми обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$, описані у відповідних підрозділах.

Algorithm 1: Алгоритм симуляції

Data: $N_A, N_P, \phi, \Delta, \Pr(\mathcal{A}), \Pr(\mathcal{P})$

$\mathbf{A} \leftarrow \{\}$;

$\mathbf{P} \leftarrow \{\}$;

for $i \in \overline{1..N_A}$ **do**

  $\mathbf{A} \leftarrow \mathbf{A} \cup \{\mathbf{a}\}, \pi(\mathbf{a}) \sim \Pr(\mathcal{A})$;

  $\tau(\mathbf{a}, 0) \leftarrow 0$

end

for $i \in \overline{1..N_P}$ **do**

  $\mathbf{P} \leftarrow \mathbf{P} \cup \{\mathbf{p}\}, \pi(\mathbf{p}) \sim \Pr(\mathcal{P})$;

end

for $t \in T$ **do**

  for $\mathbf{a} \in \mathbf{A}$ **do**

    $\xi(\mathbf{a}, t+1) \leftarrow \overline{s} \sim \mathcal{P}_\Delta(\mathbf{a}, t)$;

    $\pi(\mathbf{a}, t+1) \leftarrow \upsilon_a(\omega, \pi(\mathbf{a}, t))$;

    if $\xi(\mathbf{a}, t+1) \neq \xi(\mathbf{a}, t)$ **then**

      $\tau(\mathbf{a}, t+1) \leftarrow 1$

    else

      $\tau(\mathbf{a}, t+1) \leftarrow \tau(\mathbf{a}, t) + 1$

    end

  end

  for $\mathbf{p} \in \mathbf{P}$ **do**

    $\pi(\mathbf{p}, t+1) \leftarrow \upsilon_p(\omega, \pi(\mathbf{p}, t))$;

  end

end

2.1.2 Тривіальний алгоритм

У алгоритмі 2 наведений опис тривіального алгоритму обчислення $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s)$, проблема якого, як зазначено вище, полягає у двох вкладених циклах по множині агентів, що й робить складність алгоритму квадратичною від кількості агентів.

Algorithm 2: Алгоритм обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$ тривіальний

Data: $\mathbf{A}, \mathbf{P}, \mathbf{H}, \mathbf{S}, \Phi_t, \Phi_p, t \in \mathbf{T}, \omega \in \Omega,$

Result: $\mathcal{P}_\Delta(\mathbf{a}, t)$

for $\mathbf{a} \in \mathbf{A}$ **do**

  for $s \in \mathbf{S}$ **do**

    $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s) \leftarrow \Phi_p(\omega, s, \pi(\mathbf{a}, t), \tau(\mathbf{a}, t))$

  end

  for $h \in \mathbf{H}$ **do**

    $\mathbf{p} \leftarrow \phi(\omega, \pi(\mathbf{a}, t), \xi(\mathbf{a}, t), t, h)$

    for $d \in \mathbf{A}$ **do**

      if $\mathbf{p} = \phi(\omega, \pi(\mathbf{d}, t), \xi(\mathbf{d}, t), t, h)$ **then**

        for $s \in \mathbf{S}$ **do**

          $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s) \leftarrow 1 - (1 - \Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s))(1 - \Phi_t(\omega, s, \pi(\mathbf{a}, t), \pi(\mathbf{d}, t), \pi(\mathbf{p}, t)))$

        end

      end

    end

  end

  $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = \xi(\mathbf{a}, t)) \leftarrow 1 - \sum_{s \in \mathbf{S}, s \neq \xi(\mathbf{a}, t)} \Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s)$

end

2.1.3 Алгоритм з групуванням

При реалізації моделей реального світу справедливим є наступне твердження:

$$|\mathbf{A}| \propto |\mathbf{P}| \tag{2.13}$$

Це можна пояснити тим, що, наприклад, середня кількість жителів однієї квартири не залежить від кількості людей у модельованому місті, країні тощо (насправді ця залежність може існувати через різний рівень життя, або інші соціодемографічні показники у країнах з різною кількістю населення, але це питання лежить поза межами цього дослідження). Нехай, модель містить множину типів місць, кількість представників якої пропорційна кількості агентів з коефіцієнтом k_i, яку у кожен часовий крок відвідують x_i агентів із довільного ймовірнісного розподілу X_i. Тоді, у разі групування агентів, що були у контакті, в окремі групи та виконання ітерування їх пар всередині таких груп, робить обчислювальну складність алгоритму рівною:

$$O(g_T(n) + \mathbb{E}[\sum_i nk_i x_i^2]) = O(g_T(n) + n\sum_i k_i \mathbb{E}[x_i^2]) = O(g_T(n) + n), \tag{2.14}$$

де $g_T(n)$ — обчислювальна складність самого алгоритму групування.

Обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$ методом групування описане в алгоритмі 3, де sus та inf — предикати, задані у рівнянні 2.15.

$$\begin{aligned} \text{sus}(a,t) =& \exists \omega \in \Omega, \exists s \in \mathbf{S}, s \neq \xi(a,t), \exists d \in \mathcal{A}, \\ & \exists p \in \mathcal{P} : \Phi_t(\omega, s, \pi(a,t), d, p) > 0 \\ \text{inf}(a,t) =& \exists \omega \in \Omega, \exists s \in \mathbf{S}, s \neq \xi(r,t), \exists r \in \mathcal{A}, \\ & \exists p \in \mathcal{P} : \Phi_t(\omega, s, r, \pi(a,t), p) > 0 \end{aligned} \tag{2.15}$$

Предикат sus показує чи є певний агент у певний час сприйнятливим до захворювання, inf - заразливим. І хоча обчислення значень цих предикатів за

означенням не є практичним, у разі задання функції Φ_t через множину правил (рівняння 2.8), воно може бути значно спрощене. Для цього, введемо наступні позначення:

$$\begin{aligned} \hat{\text{sus}}(a,t) &= \exists f, \exists s_d, s \in \mathbf{S} : ((\xi(a,t), s_d, s), f) \in \mathbf{R_t} \\ \hat{\text{inf}}(a,t) &= \exists f, \exists s_r, s \in \mathbf{S} : ((s_r, \xi(a,t), s), f) \in \mathbf{R_t} \end{aligned} \tag{2.16}$$

Очевидно, що $\neg\hat{\text{sus}}(a,t) \Rightarrow \neg\text{sus}(a,t)$ і $\neg\hat{\text{inf}}(a,t) \Rightarrow \neg\text{inf}(a,t)$. Це означає, що sus та inf можуть бути грубо задані як $\hat{\text{sus}}$ та $\hat{\text{inf}}$ відповідно.

Алгоритм 3 передбачає групування агентів одразу на дві групи — за сприятливістю/заразливістю. Таке розбиття дозволяє зменшити кількість пар розглянутих агентів, адже оминає ті пари, у яких, наприклад, обидва агенти не є заразливими та відповідно не можуть вплинути на стан один одного.

Обчислювальна та просторова алгоритмічні складності алгоритму групування напряму залежать від структури даних, що лежить в його основі. Для успішної реалізації алгоритму, ця структура має підтримувати наступні операції:

- Для пари агент-місце (a, p) додати її у колекцію;
- Для місця p повернути усіх агентів a_i, для яких пари (a_i, p) були доданими у колекцію раніше;
- Очистити колекцію.

Окрім цього, наявні наступні обмеження:

- Кількість пар (a, p) попередньо відома та дорівнює $|\mathbf{H}| \cdot |\mathbf{A}|$;
- Кількість ключів попередньо відома та не перевищує $|\mathbf{P}| \cdot |\mathbf{H}|$;
- Максимальна кількість значень за одним ключем дорівнює $|\mathbf{A}|$.

Такою структурою даних є мультисловник (hashmap, hashdictionary), що є розширенням асоціативного масиву для випадку збереження декількох значень за одним ключем. Являючись абстрактним типом даних, мультисловник може бути реалізований декількома різними способами, що мають спільну рису: поділ структури на два компоненти. Першою компонентою є структура даних, що

Algorithm 3: Алгоритм обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$ з групуванням

Data: $\mathbf{A}, \mathbf{P}, \mathbf{H}, \mathbf{S}, \Phi_t, \Phi_p, t \in \mathbf{T}, \omega \in \Omega,$

Result: $\mathcal{P}_\Delta(\mathbf{a}, t)$

$G_s \leftarrow \{\emptyset | \forall (p, h) \in \mathbf{P} \times \mathbf{H}\}$

$G_i \leftarrow \{\emptyset | \forall (p, h) \in \mathbf{P} \times \mathbf{H}\}$

for $\mathbf{a} \in \mathbf{A}$ **do**

  for $s \in \mathbf{S}$ **do**

    $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s) \leftarrow \Phi_p(\omega, s, \pi(\mathbf{a}, t), \tau(\mathbf{a}, t))$

  end

  for $h \in \mathbf{H}$ **do**

    $\mathbf{p} \leftarrow \phi(\omega, \pi(\mathbf{a}, t), \xi(\mathbf{a}, t), t, h)$

    if $\mathrm{sus}(\mathrm{a}, t)$ **then**

      $G_{s,(\mathbf{p},h)} \leftarrow G_{s,(\mathbf{p},h)} \cup \{\mathbf{a}\}$

    end

    if $\mathrm{inf}(\mathrm{a}, t)$ **then**

      $G_{i,(\mathbf{p},h)} \leftarrow G_{i,(\mathbf{p},h)} \cup \{\mathbf{a}\}$

    end

  end

end

for $(\mathbf{p}, h) \in \mathbf{P} \times \mathbf{H}$ **do**

  for $(\mathbf{r}, \mathbf{d}) \in G_{s,\mathbf{p},h} \times G_{i,\mathbf{p},h}$ **do**

    for $s \in \mathbf{S}$ **do**

      $\Pr(\mathcal{P}_\Delta(\mathbf{r}, t) = s) \leftarrow 1 - (1 - \Pr(\mathcal{P}_\Delta(\mathbf{r}, t) = s))(1 - \Phi_t(\omega, s, \pi(\mathbf{r}, t), \pi(\mathbf{d}, t), \pi(\mathbf{p}, t)))$

    end

  end

end

for $\mathbf{a} \in \mathbf{A}$ **do**

  $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = \xi(\mathbf{a}, t)) \leftarrow 1 - \sum_{s \in \mathbf{S}, s \neq \xi(\mathbf{a}, t)} \Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s)$

end

відображає ключ у посилання на контейнер, що містить список значень; другою — тип цього контейнера. Так, наприклад, першою структурою даних може бути геш-таблиця [78], а другою — зв'язний список [79].

Виходячи с цього, обчислювальна алгоритмічна складність g_T може бути задана як:

$$\begin{aligned} g_T(n) &= g_{T1}(n) + g_{T2}(n) \\ g_{T1}(n) &= O(n \cdot \text{ind}_1(n)\text{ins}_2(n)) \\ g_{T2}(n) &= O(\text{itr}_1(n)\text{itr}_2(n)), \end{aligned} \tag{2.17}$$

де ind_1 — алгоритмічна складність пошуку за індексом першої структури даних, ins_2 — алгоритмічна складність вставки у кінець другої структури даних, $\text{itr}_1, \text{itr}_2$ — алгоритмічна складність ітерування елементів першої та другої структур даних відповідно.

Виходячи з зазначених вище обмежень та факту того, що усі ключі можуть бути пронумеровані від 1 до $|\mathbf{P}| \cdot |\mathbf{H}|$, очевидними кандидатами на першу структуру даних є статичний масив та зв'язний список оскільки для них є справедливим: $\text{ind}_1(n) = O(1), \text{itr}_1(n) = O(n)$. Проте, завдяки використанню кешу центрального процесора, на практиці використання статичного масиву є ефективнішим (рисунок 2.14).

Окрім реалізації мультисету за допомогою контейнеру контейнерів може бути використаний підхід, оснований на геш-таблицях. У такому разі, реалізація містить дві структури даних, перша з яких відображує значення ключа у кількість елементів за цим ключем, що збережені у структурі, а друга — ключ та індекс елемента у його значення. Схематичне зображення наведено у рисунку 2.15.

Такий підхід є доцільним, адже операції додавання та отримання значення за ключем у таку структуру будуть мати алгоритмічну складність рівну $O(1)$. Це є справедливим через те, що аналогічні операції щодо геш-таблиці мають також константну складність (у разі відсутності колізій) [80].

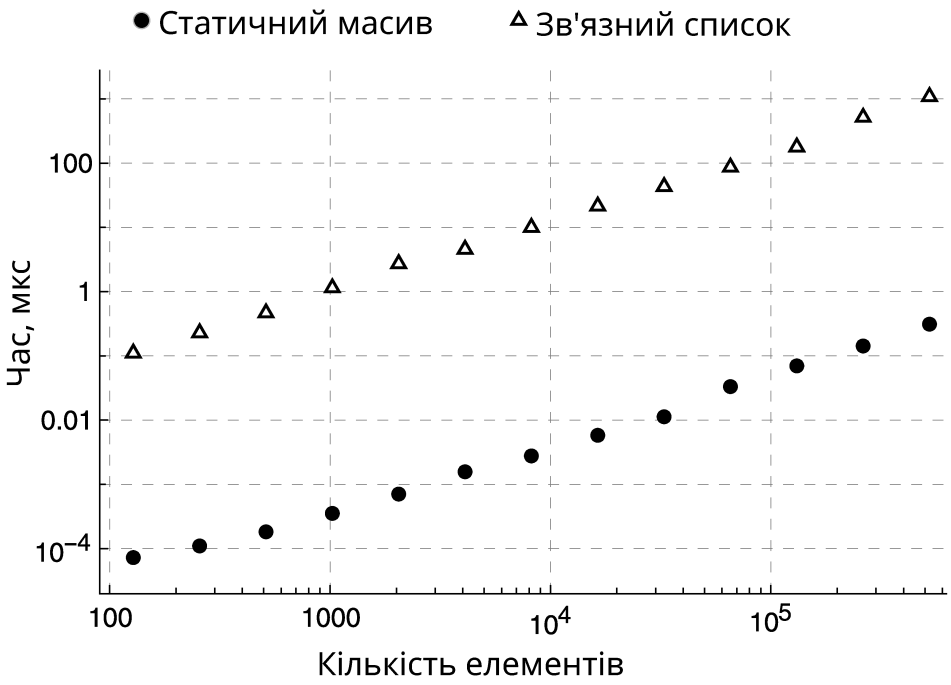

Рис. 2.14: Порівняння швидкості ітерування контейнерів. Компілятор cargo 1.56.0, оптимізації компілятора вимкнені

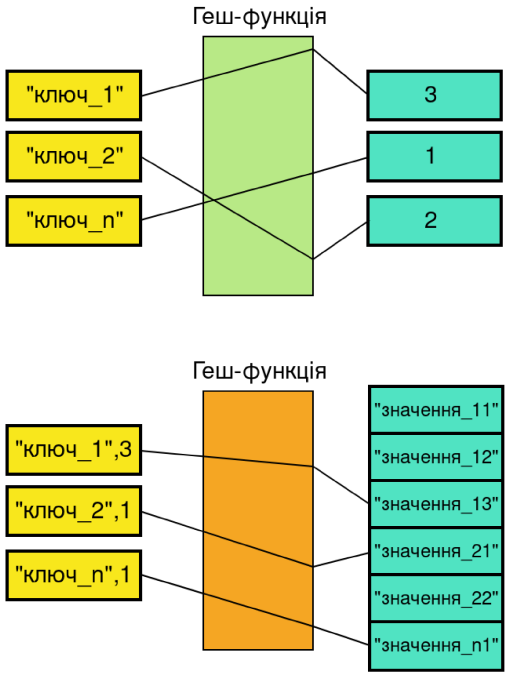

Рис. 2.15: Схематичне зображення реалізації мультисету за допомогою геш-таблиць

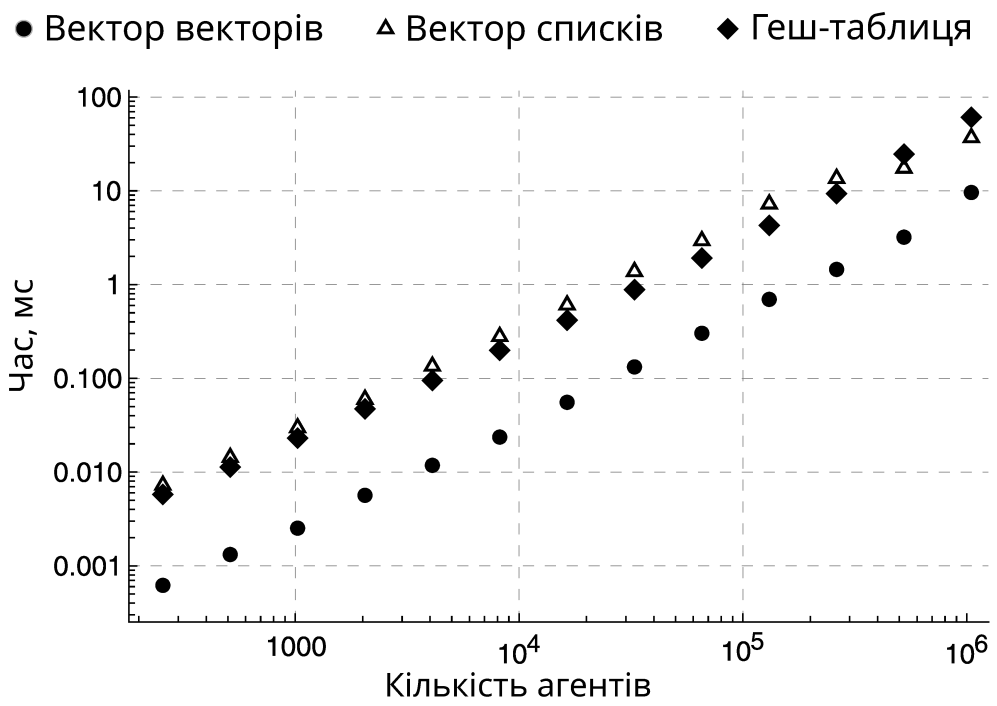

Рис. 2.16: Час роботи алгоритму групування для середнього відношення кількості агентів до кількості місць рівного 16:1

Для оцінки швидкодії кожного підходу був проведений ряд експериментів, що передбачали обчислення часу виконання алгоритму групування для кожного підходу за різної кількості агентів та відношення кількості агентів до кількості місць. Результати наведені у рисунках 2.16, 2.17, 2.18.

З результатів видно, що підхід до реалізації мультисету через вектор векторів є значно ефективнішим. Це можна пояснити, що за оптимально налаштованого розкладу зміни розміру вектора можна звести кількість алокації нової пам'яті до мінімуму. І хоча неможливо передбачити кількість елементів за кожним ключем наперед не маючи щодо цього апріорної інформації, у разі використання мультисету для урахування відвідувань агентами місць у рамках інфекційного моделювання, можна покласти, що ця кількість залишається приблизно однаковою. Тобто:

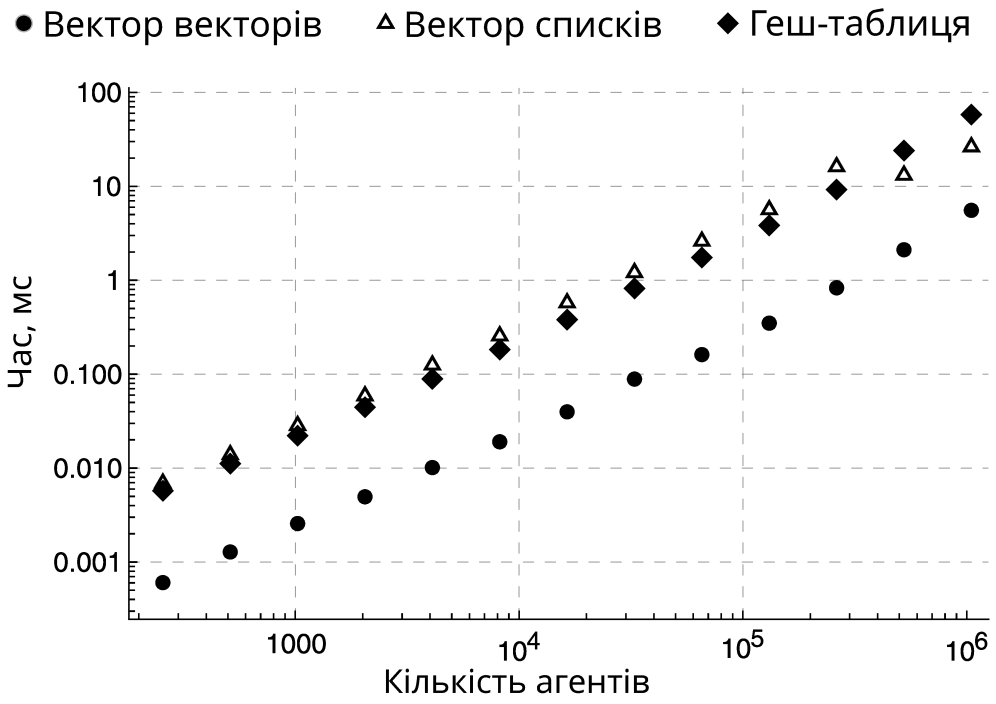

Рис. 2.17: Час роботи алгоритму групування для середнього відношення кількості агентів до кількості місць рівного 256:1

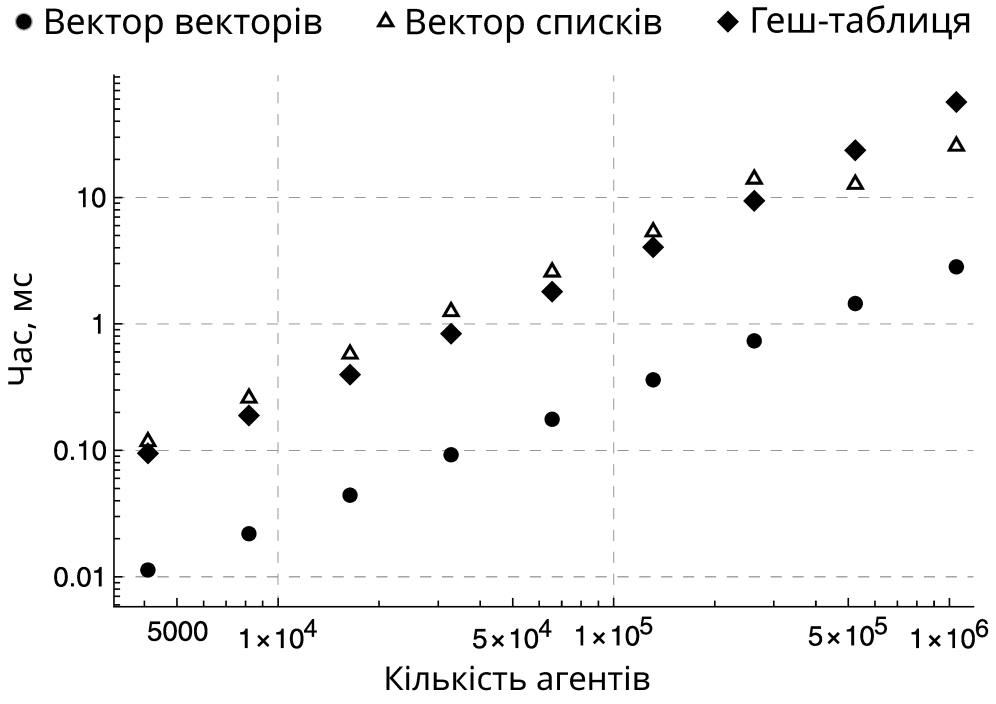

Рис. 2.18: Час роботи алгоритму групування для середнього відношення кількості агентів до кількості місць рівного 4096:1

$$\forall \mathbf{p} \in \mathbf{P}, \forall t \in \mathbf{T}, \forall h \in \mathbf{H} : \sum_{\mathbf{a}} \mathbb{1}(\mathbf{p} = \phi(\omega, \pi(\mathbf{a}, t), \xi(\mathbf{a}, t), t, h)) \approx \sum_{\mathbf{a}} \mathbb{1}(\mathbf{p} = \phi(\omega, \pi(\mathbf{a}, t+1), \xi(\mathbf{a}, t+1), t+1, h)) \tag{2.18}$$

Очевидно, що у реальному світі це твердження загалом не є вірним. Наприклад, логічно припустити, що кількість відвідувачів торговельно-розважальних центрів значно збільшується з п'ятниці на суботу у країнах, де робоча неділя завершується у п'ятницю. Проте, це може бути частково вирішеним, якщо утримувати розмір буферів рівним щонайменш останній кількості елементів, помноженій на певну константу.

У загальному випадку, можна досягнути мінімальної кількості алокацій пам'яті рівній нулю якщо встановити розмір кожного буфера рівним кількості агентів. Проте, через припущення 2.13, кількість самих буферів є пропорційною кількості агентів. Це робить просторову складність такого алгоритму рівною $O(n^2)$, що є недопустимим.

Окремо слід розглянути випадок, у разі якого ймовірність передачі захворювання в момент контакту двох агентів не залежить від параметрів донора (окрім стану захворювання), що може бути виражене наступним твердженням:

$$\forall s_a, s_d, s \in \mathbf{S}, \forall \omega \in \Omega, \forall a_r, a_1, a_2 \in \mathcal{A}, \forall p \in \mathcal{P}, \nexists((s_a, s_d, s), f) \in \mathbf{R}_t : f(\omega, s, s_a, s_d, a_r, a_1, p) \neq f(\omega, s, s_a, s_d, a_r, a_2, p) \tag{2.19}$$

І хоча, не існує передумов вважати що це твердження вірне для реального світу, більшість розглянутих робіт не розглядає випадок коли воно не є таким. У такому разі, алгоритм обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$ може бути значно спрощений. Спрощена версія наведена в алгоритмі 4. Як можна побачити із формулювання алгоритму, основна його частина складається з двох послідовних циклів по множині агентів, що робить його алгоритмічну складність рівною $O(n)$.

Окрім підходу, що передбачає обрахунок $\mathcal{P}_\Delta(\mathbf{a}, t)$ існує дискретний підхід до симуляції, кроки якого узагальнені в алгоритмі 5.

Algorithm 4: Алгоритм обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$ спрощений

Data: $\mathbf{A}, \mathbf{P}, \mathbf{H}, \mathbf{S}, \Phi_t, \Phi_p, t \in \mathbf{T}, \omega \in \Omega,$

Result: $\mathcal{P}_\Delta(\mathbf{a}, t)$

for $\mathbf{p} \in \mathbf{P}$ **do**
  for $h \in \mathbf{H}$ **do**
    for $(s_1, s_2) \in \mathbf{S}^2$ **do**
      $I(\mathbf{p}, h, s_1, s_2) \leftarrow 0$
    end
  end
end
for $\mathbf{a} \in \mathbf{A}$ **do**
  if $\inf(\mathbf{a}, t)$ **then**
    for $h \in \mathbf{H}$ **do**
      $\mathbf{p} \leftarrow \phi(\omega, \pi(\mathbf{a}, t), \xi(\mathbf{a}, t), t, h)$
      for $(s_1, s_2) \in \mathbf{S}^2$ **do**
        $I(\mathbf{p}, h, s_1, s_2) \leftarrow$
        $1 - (1 - I(\mathbf{p}, h, s_1, s_2))(1 - \Phi_t(\omega, s_2, s_1, \pi(\mathbf{a}, t), \pi(\mathbf{p}, t)))$
      end
    end
  end
end

for $\mathbf{a} \in \mathbf{A}$ **do**

  for $s \in \mathbf{S}$ **do**

    $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s) \leftarrow \Phi_p(\omega, s, \pi(\mathbf{a}, t), \tau(\mathbf{a}, t))$

  end

  if $\text{sus}(\mathbf{a}, t)$ **then**

    for $h \in \mathbf{H}$ **do**

      $\mathbf{p} \leftarrow \phi(\omega, \pi(\mathbf{a}, t), \xi(\mathbf{a}, t), t, h)$

      for $s \in \mathbf{S}$ **do**

        $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s) \leftarrow 1 - (1 - \Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s))(1 - I(\mathbf{p}, h, \xi(\mathbf{a}, t), s))$

      end

    end

  end

  $\Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = \xi(\mathbf{a}, t)) \leftarrow 1 - \sum_{s \in \mathbf{S}, s \neq \xi(\mathbf{a}, t)} \Pr(\mathcal{P}_\Delta(\mathbf{a}, t) = s)$

end

Algorithm 5: Алгоритм симуляції дискретний

Data: $\mathbf{A}, \mathbf{P}, \mathbf{H}, \mathbf{S}, \Phi_t, \Phi_p, t \in \mathbf{T}, \omega \in \Omega,$

Result: $\mathcal{P}_\Delta(\mathbf{a}, t)$

for $\mathbf{a} \in \mathbf{A}$ **do**

$\tau(\mathbf{a}, t+1) \leftarrow \tau(\mathbf{a}, t) + 1$

$\hat{s} \sim \mathcal{S} : \forall s \in \mathbf{S}, \Pr(\mathcal{S} = s) = \Phi_p(\omega, s, \pi(\mathbf{a}, t), \tau(\mathbf{a}, t))$

if $\hat{s} = \xi(\mathbf{a}, t)$ **then**

$\mathbf{p} \leftarrow \phi(\omega, \pi(\mathbf{a}, t), \xi(\mathbf{a}, t), t, h)$

for $\mathbf{d} \in \mathbf{A}$ **do**

if $\mathbf{p} \neq \phi(\omega, \mathbf{d}, \xi(\mathbf{d}, t), t, h)$ **then**

continue

end

$\hat{s} \sim \mathcal{S} : \forall s \in \mathbf{S}, \Pr(\mathcal{S} = s) = \Phi_t(\omega, s, \pi(\mathbf{a}, t), \pi(\mathbf{d}, t), \pi(\mathbf{p}, t))$

if $\hat{s} \neq \xi(\mathbf{a}, t)$ **then**

$\xi(\mathbf{a}, t+1) \leftarrow \hat{s}$

$\tau(\mathbf{a}, t+1) \leftarrow 0$

break

end

end

else

$\xi(\mathbf{a}, t+1) \leftarrow \hat{s}$

$\tau(\mathbf{a}, t+1) \leftarrow 0$

end

end

До недоліків дискретного алгоритму симуляції можна віднести квадратичну обчислювальну складність та неоднозначність у питанні обчислення стану кожного агенту у наступний крок симуляції. Адже є невизначеним порядок застосування Φ_t та Φ_p. Проте, просторова складність рівна $O(1)$ робить його більш придатним для використання у випадку обмежених просторових ресурсів.

У рисунку 2.19 наведене порівняння швидкодії вищеназваних алгоритмів обчислення $\mathcal{P}_{\Delta}(\mathbf{a}, t)$ для заданої тестової епідеміологічної моделі. Виходячи з отриманих практичних результатів можна зробити висновок, що для підвищення швидкодії роботи моделі алгоритм 4 має бути використаним якщо є справедливими умови його застосування. В іншому разі має бути використаним алгоритм 3.

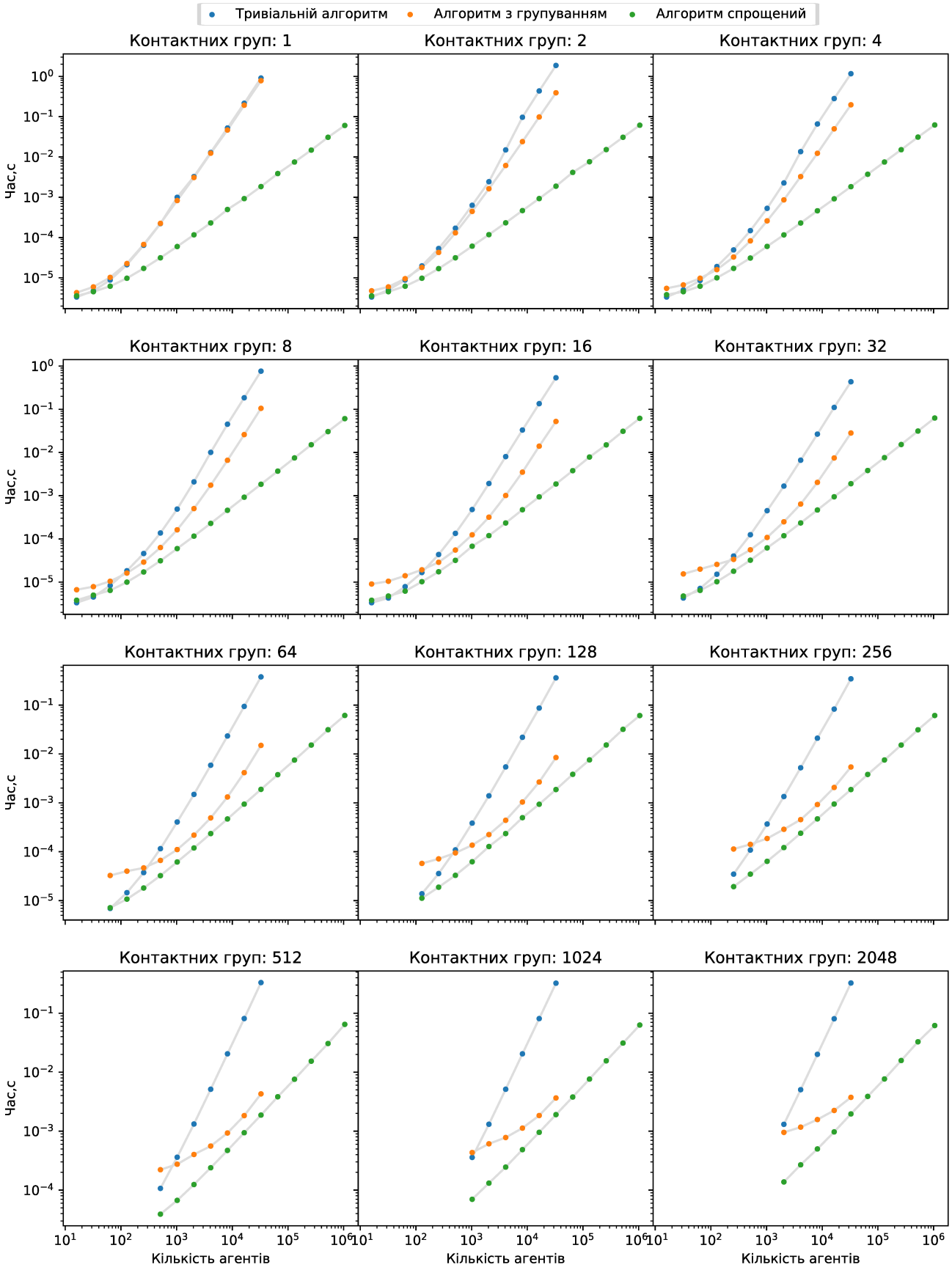

Рис. 2.19: Час обрахунку кроку симуляції тестової епідеміологічної моделі для різних алгоритмів обчислення $\mathcal{P}_\Delta(\mathbf{a}, t)$

2.2 Формальна мова опису агентно-орієнтованої моделі розповсюдження інфекційних захворювань

2.2.1 Теоретичні положення

Окремою категорією підходів до реалізації моделей є використання так званих предметно-орієнтованих мов програмування. Їм, як мовам програмування, характерна лежача в основі формальна граматика. Формальна граматика є способом опису формальної мови [61] і складається з наступних компонент:

- Кінцевої множини терміналів Σ;
- Кінцевої множини нетерміналів N;
- Початкового символу S;
- Кінцевої множини правил продукції P.

Терміналами називаються символи формальної граматики що є кінцевими, для яких немає жодного правила із P, де ці символи стояли б у лівій частині. Найчастіше терміналами являються букви, цифри, спеціальні символи друку.

Нетермінали – символи формальної граматики, що, на відміну від терміналів, не є кінцевими. У разі коректно сформульованої формальної граматики, кожному нетерміналу відповідає підмножина правил, де цей нетермінал стоїть у лівій частині. Нетермінали найчастіше описують більш загальні концепції такі як “число”, “слово”, “речення”. Серед множини нетерміналів визначається окремий символ P, що називається початковим. Цей символ позначає найабстрактнішу концепцію, наприклад “текст” або “лістинг програми”.

Правило продукції $\alpha \rightarrow \beta$ характеризується парою (α, β), що являються елементами множини рядків словника $V = \Sigma \cup N$. За цим правилом, рядок символів α може бути замінений на β.

В залежності від типів правил продукції за ієрархією Чомскі [81] формальні граматики поділяються на чотири групи:

- Довільна формальна граматика (тип 0) - формальна граматика, за якої $\forall(\alpha \rightarrow$

$\beta) \in P : \alpha \in V^+, \beta \in V^*$;

• Контекстно-залежна формальна граматика (тип 1) - формальна граматика, за якої $\forall(\alpha A\beta \to \alpha\gamma\beta) \in P : \alpha, \beta \in V^*, \gamma \in V^+, A \in N$;

• Контекстно-вільна формальна граматика (тип 2) - формальна граматика, за якої $\forall(A \to \gamma) \in P : A \in N, \gamma \in V^*$;

• Регулярна формальна граматика (тип 3) - формальна граматика, за якої $\forall(A \to B\gamma) \in P : A, B \in N, \gamma \in V^*$ для ліволінійних граматик, або $\forall(A \to \gamma B) \in P : A, B \in N, \gamma \in V^*$ для праволінійних.

Тут V^* — множина усіх рядків словника V, V^+ — множина усіх непустих рядків словника V. Найбільший інтерес у рамках цієї роботи представляють контекстно-вільні формальні граматики, адже вони використовуються для опису синтаксису більшості мов програмування.

Для наочного представлення формальних граматик типу 2 найчастіше використовується нотація Бекуса-Наура [82] (зазвичай її розширена версія [83]) та графічний варіант її представлення — синтаксична або залізнична діаграма (англ. railroad diagram). Опис формальної мови нотацією Бекуса-Наура складається з множини виразів, кожному з яких відповідає одне чи декілька правил продукції зі спільною лівою частиною. Ці вирази мають вигляд:

<символ> ::= __вираз__

Тут “символ” — нетермінал, що стоїть у лівій частині правил продукції, що визначається; “вираз” - рядок описує агреговані праві частині правил продукції. Нотація Бекуса-Наура підтримує спеціальний оператор альтернативи ”|“, що спрощує запис правил продукції. Наприклад, нехай граматика деякої формальної мови представлена наступними правилами:

$$\begin{aligned} A &\to a \\ A &\to ab \end{aligned} \tag{2.20}$$

Використовуючи оператор альтернативи, така граматика може бути представлена одним виразом:

<A> ::= “a” | “a” “b”

Нотація Бекуса-Наура не передбачає окремого задання множин терміналів та нетерміналів. Натомість нетерміналами вважаються символи, що були у лівій частині правил, терміналами — усі інші. Стартовим символом *S* вважається нетермінал, заданий у першому правилі.

Розширена нотація Бекуса-Наура відрізняється від оригінальної перш за все операторами повтору, опціонального включення та групування. Оператор повтору “{...}” описує правило, за якого його аргумент може повторюватись декілька разів. Наприклад:

слово = буква, {буква}

є еквівалентним:

<слово> ::= <буква> | <слово> <буква>

Оператор опціонального включення “[...]” дозволяє описувати правило, за якого наявність певного підрядка є опціональною. Наприклад:

число = [“-”], {додатне число}

є еквівалентним:

<число> ::= “-” <додатне число> | <додатне число>

Оператор групування “(...)” дозволяє вказувати пріоритет для інших операторів по аналогії з дужками у математичних виразах.

Розширена нотація Бекуса-Наура є зручним способом опису граматики формальних мов, більш компактною за звичайний її варіант. Окрім текстового запису, вона може бути представлена за допомогою синтаксичних діаграм, які також будуть використані дали у цій роботі.

2.2.2 Формальна граматика мови опису агентно-орієнтованої моделі розповсюдження інфекційних захворювань

Множина терміналів Σ складається з наступних символів:

“!=”, “$”, “(”, “)”, “*”, “+”, “,”, “-”, “->”, “.”, “/”, “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”‘, “8”, “9”, “:”, “<”, “<-”, “<=”, “=”, “==”, “>”, “>=”, “A”, “B”, “C”, “D”, “E”, “F”, “G”, “H”, “I”, “J”, “K”, “L”, “M”, “N”, “O”, “P”, “Q”, “R”, “S”, “T”, “U”, “V”, “W”, “X”, “Y”, “Z”, “[”, “]”, “^”, “_”, “a”, “b”, “c”, “d”, “e”, “f”, “g”, “h”, “i”, “j”, “k”, “l”, “m”, “n”, “o”, “p”, “q”, “r”, “s”, “t”, “u”, “v”, “w”, “x”, “y”, “z”, “{”, “|”, “}”, “~”.

Множина нетерміналів N складається з наступних символів:

“program”, “declaration”, “function_declaration”, “param_declaration”, “distribution_declaration”, “distribution_disc_declaration”, “distribution_cont_declaration”, “agent_declaration”, “struct_declaration”, “infection”, “infection_transmission_rule”, “infection_progress_rule”, “transition”, “expression”, “digit”, “natural”, “integer”, “float”, “number”, “id”, “letter”.

Стартовим символом S є символ ”program“.

Лістинг програми розробленої мови складається із двох елементів: оголошень глобальних сутностей та опис параметрів модельованого інфекційного захворювання. Серед оголошень обов’язковим є оголошення агенту. Синтаксична діаграма символу “program” наведена у рисунку 2.20.

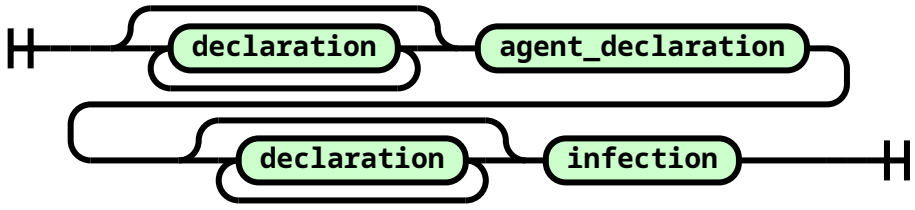

Рис. 2.20: Опис символу “program”

Оголошенням глобальних сутностей може бути оголошення параметра моделі, ймовірнісного розподілу, функції, або структури. Синтаксична діаграма

символу “declaration” наведена у рисунку 2.21.

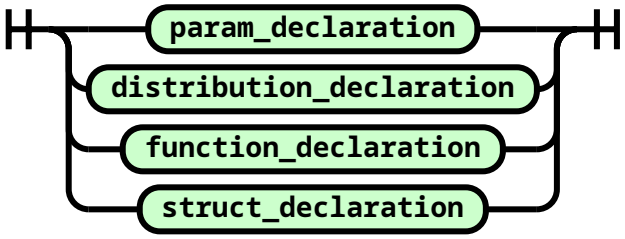

Рис. 2.21: Опис символу “declaration”

Оголошення функції проводиться через задання її назви, списку аргументів та значення. Синтаксична діаграма символу “function_declaration” наведена у рисунку 2.22.

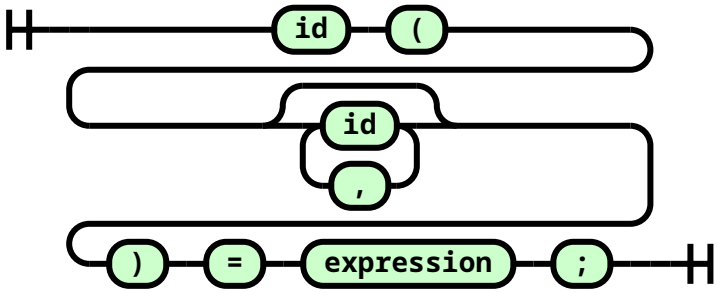

Рис. 2.22: Опис символу “function_declaration”

Параметром моделі може бути будь-яке скалярне значення, задане константою, або взяте з ймовірнісного розподілу. Синтаксична діаграма символу “param_declaration” наведена у рисунку 2.23.

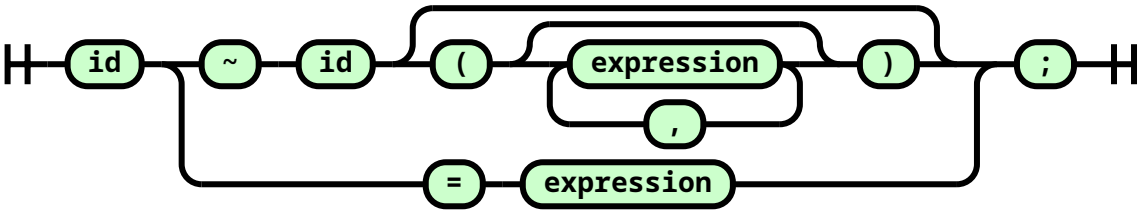

Рис. 2.23: Опис символу “param_declaration”

Синтаксис мови передбачає окремі символи задання дискретного та неперервного ймовірнісного розподілу. Синтаксична діаграма символу “distribution_declaration” наведена у рисунку 2.24.

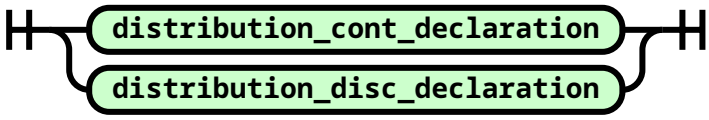

Рис. 2.24: Опис символу “distribution_declaration”

Оголошення неперервного ймовірнісного розподілу відбувається через задання його квантильної функції [84]. Синтаксична діаграма символу “distribution_cont_declaration” наведена у рисунку 2.25.

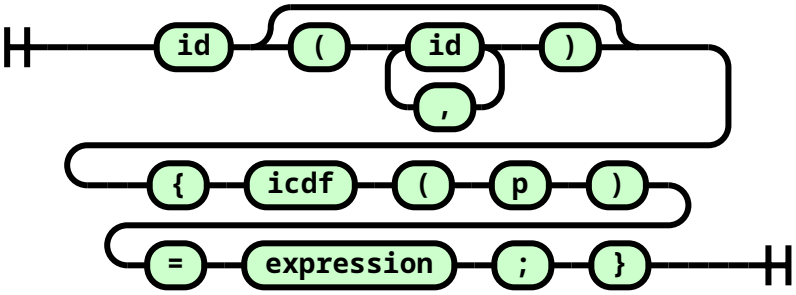

Рис. 2.25: Опис символу “distribution_cont_declaration”

Оголошення структурного типу відбувається через задання його назви, типу структури, кількості елементів, набору внутрішніх параметрів, методів та правил переходу. Синтаксична діаграма символу “struct_declaration” наведена у рисунку 2.26.

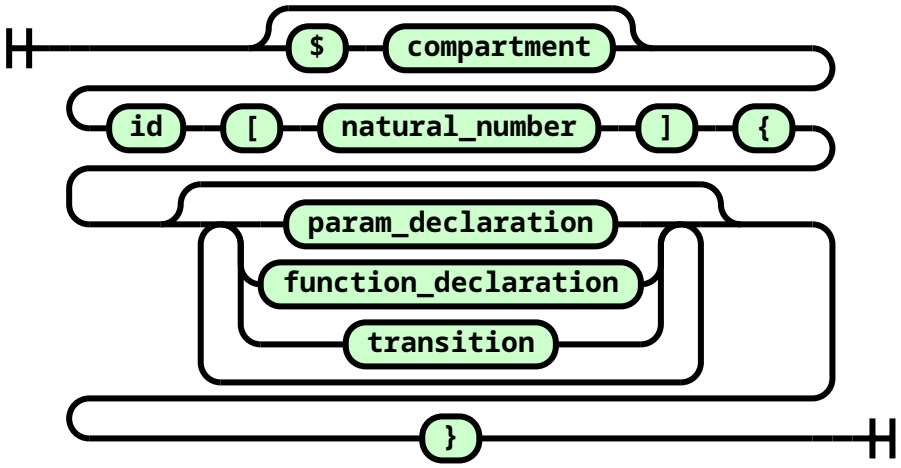

Рис. 2.26: Опис символу “struct_declaration”

Особливим випадком декларації структурного типу є оголошення структури агенту. У цьому випадку серед її методів є обов’язковим наявність методу “schedule”. Синтаксична діаграма символу “agent_declaration” наведена у рисунку 2.27.

Правила еволюції внутрішніх параметрів можуть бути задані при декларації структури за допомогою символу “transition”. Синтаксична діаграма наведена у рисунку 2.28.

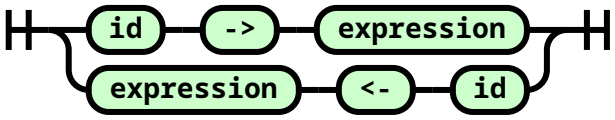

Рис. 2.28: Опис символу “transition”

Задання параметрів інфекційного захворювання відбувається за допомогою опису трьох компонентів: списку станів із заданням початкового розподілу серед населення, списку правил розповсюдження $\mathbf{R}_t$ та списку правил розвитку $\mathbf{R}_p$. Синтаксична діаграма символу “infection” наведена у рисунку 2.29.

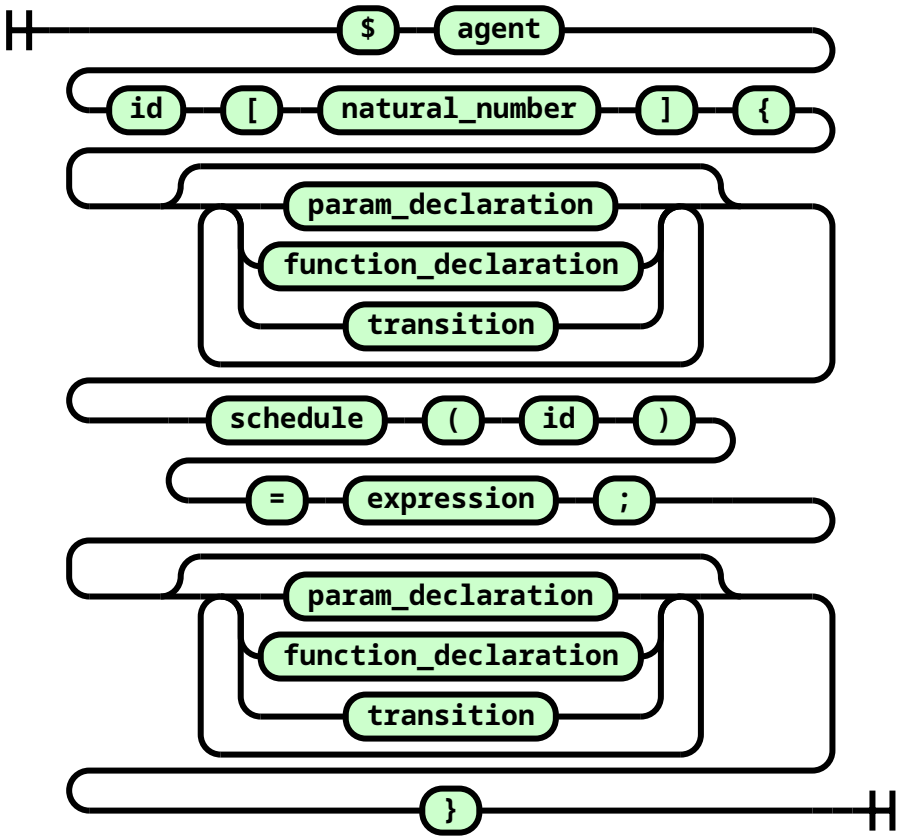

Рис. 2.27: Опис символу “agent_declaration”

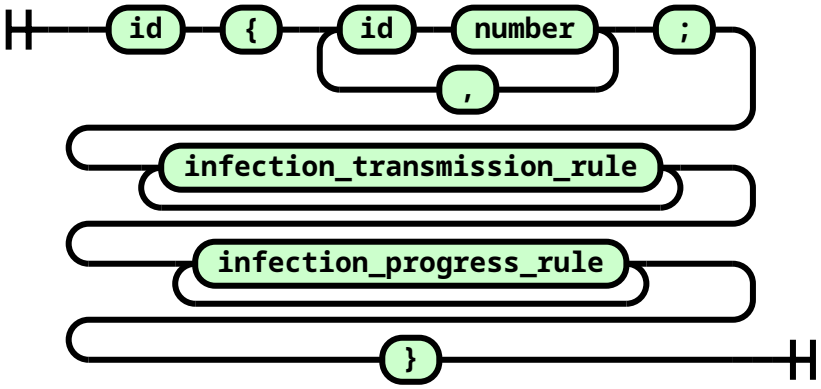

Рис. 2.29: Опис символу “infection”

Оголошення кожного правила розповсюдження $r = ((s_a, s_d, s), f) \in \mathbf{R}_t$ відбувається через задання початкового (s_a) та кінцевого (s) станів захворювання реципієнту, стану захворювання донору (s_d) та виразу, що задає ймовірність переходу (f). Синтаксична діаграма символу “infection_ transmission_ rule” на-

ведена у рисунку 2.30.

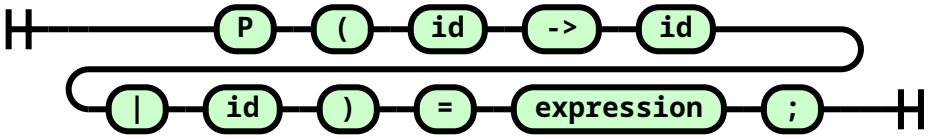

Рис. 2.30: Опис символу “infection_transmission_rule”

Оголошення кожного правила розвитку $r = ((s_a, s, \Delta t), f) \in \mathbf{R}_p$ відбувається через задання початкового (s_a) та кінцевого (s) станів агенту, виразу, що задає ймовірність переходу (f) та, за необхідності, кількості днів з останнього переходу між станами. Синтаксична діаграма символу “infection_progress_rule” наведена у рисунку 2.31.

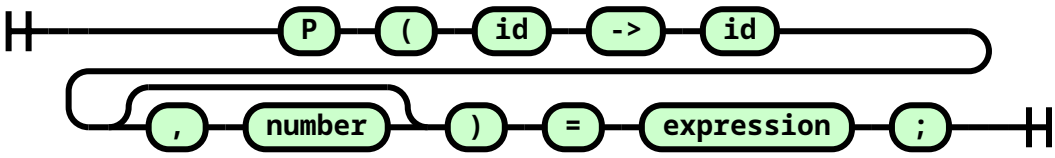

Рис. 2.31: Опис символу “infection_progress_rule”

Математичні вирази можуть бути задані за допомогою використання бінарних операторів, викликів функцій, доступу до полів структур та відображені у формальній граматиці символом “expression”. Синтаксична діаграма наведена у рисунку 2.32.

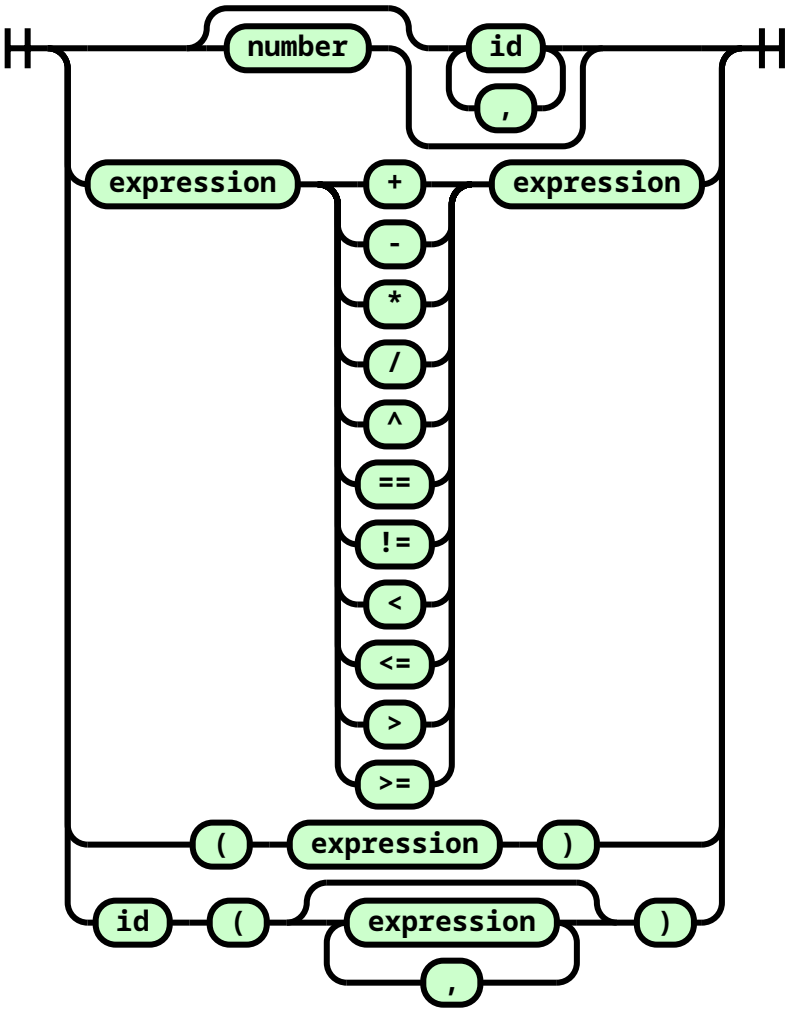

Рис. 2.32: Опис символу “expression”

Символи “digit”, “natural”, “integer”, “float” використані для задання цифр, натуральних, цілих та дійсних чисел відповідно. Синтаксичні діаграми наведені у рисунках 2.33-2.36.

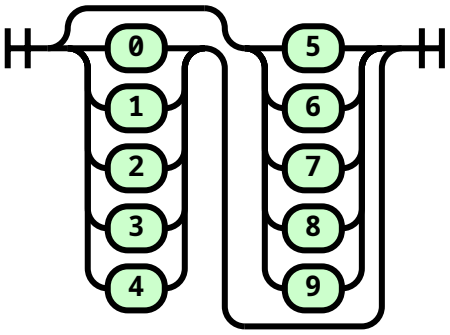

Рис. 2.33: Опис символу “digit”

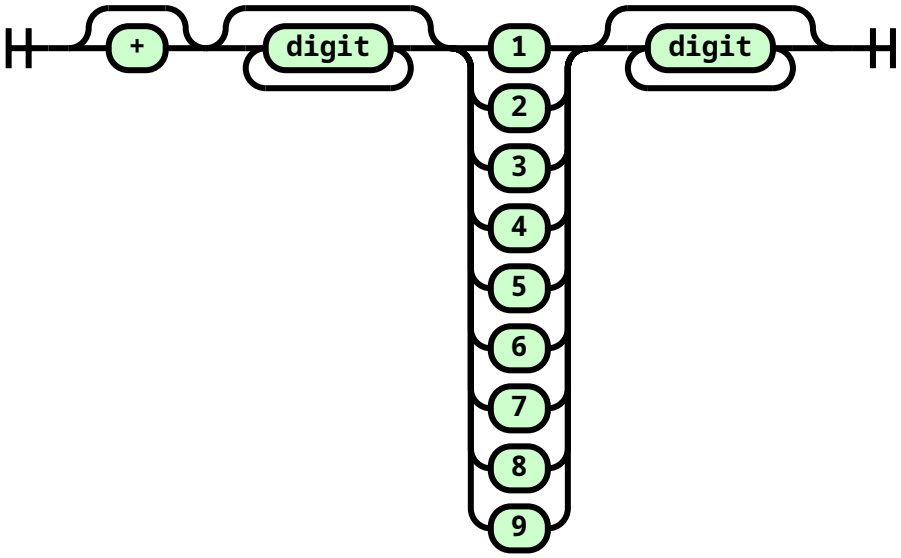

Рис. 2.34: Опис символу “natural”

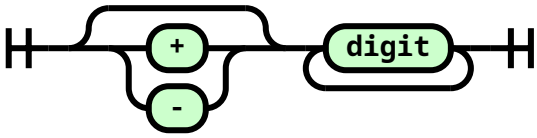

Рис. 2.35: Опис символу “integer”

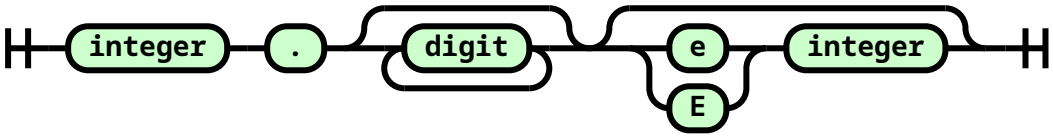

Рис. 2.36: Опис символу “float”

Для задання ідентифікаторів використовується символ “id”, що складається з букв, яким відповідає символ “letter”, цифр та терміналу “_”. Синтаксичні діаграми наведені у рисунках 2.37, 2.38.

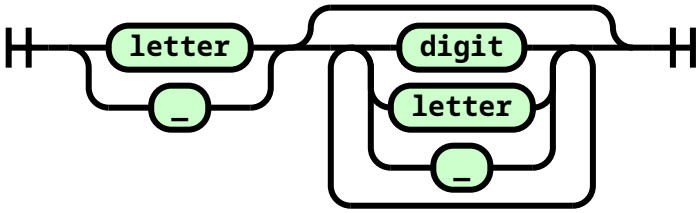

Рис. 2.37: Опис символу “id”

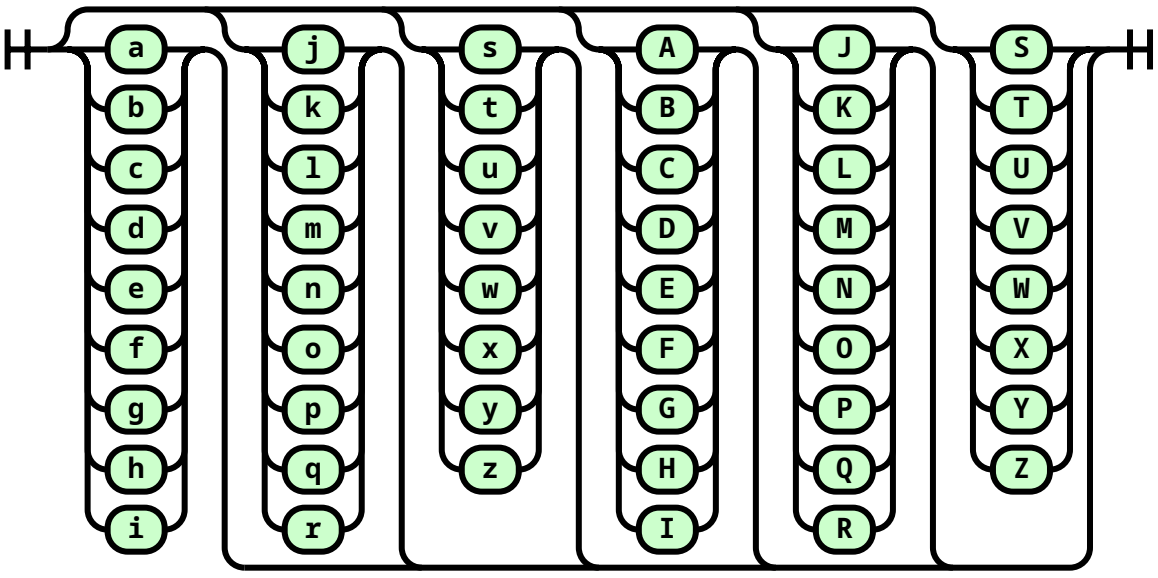

Рис. 2.38: Опис символу “letter”

Опис правил продукції розширеною нотацією Бекуса-Наура наведений у додатку Б.

ВИСНОВКИ ДО ДРУГОГО РОЗДІЛУ

У рамках другого розділу була виконана побудова загального математичного апарату епідеміологічної моделі агентного типу та розроблена граматика формальної мови її опису.

Розроблена математична модель має за основу дві множини сутностей – множину агентів та місць їх перебування. Кожна сутність є окремим об’єктом зі своїм набором значень внутрішніх параметрів, що можуть бути як статичними, так і динамічними у часі. Агенти, керуючись їх функцією розкладу можуть змінювати своє розташування у межах середовища моделювання, контактувати один з одним, призводячи до розповсюдження захворювання. Саме модельоване захворювання задається функціями розповсюдження й еволюції, що контролюють як стан захворювання агентів змінюється при контакті один з одним та із плином часу відповідно.

У рамках математичної моделі була сформульована основна складність агентного епідеміологічного моделювання, а саме урахування контактів під час обчислення розподілу майбутнього стану захворювання. Було запропоновано три алгоритми обчислення цього розподілу, проаналізована їх алгоритмічна складність. Також, для одного з них – алгоритму з групуванням було виконане дослідження оптимальної його імплементації.

З огляду на розроблену математичну модель було дано визначення формальної граматики предметно-орієнтованої мови опису агентної епідеміологічної моделі CTrace. Мова надає вбудований високорівневий інтерфейс для задання основних сутностей заданої загальної епідеміологічної моделі, а також відношень між ними. Мова підтримує задання ймовірнісних розподілів різного типу та основні операції між ними що значно спрощує задання соціодемографічної параметризації моделей.

РОЗДІЛ 3.

ПРОГРАМНІ КОМПОНЕНТИ РЕАЛІЗАЦІЇ ТА АНАЛІЗУ АГЕНТНО-ОРІЄНТОВАНОЇ МОДЕЛІ РОЗПОВСЮДЖЕННЯ ЕПІДЕМІЙ

3.1 Загальні положення

Необхідною для використання штучних формальних мов для побудови моделей є наявність транслятора — спеціалізованого програмного забезпечення, що перетворює лістинг програми однією мовою програмування у лістинг іншою мовою. Транслятори можуть бути поділені за типом результівного артефакту на інтерпретатори та компілятори. Першим характерне построкове виконання програмного коду, під час якого кожен рядок може транслюватись у деяку мову внутрішнього представлення з подальшим виконанням. З іншого боку, компілятори аналізують лістинг програмного коду цілком перед його трансляцією у вихідну мову. І хоча інтерпретаторам характерна більша гнучкість під час побудови програмного забезпечення, доступ компіляторів до всього програмного коду дозволяє виконувати оптимізацію. Це призводить до того, що виконання програм інтерпретаторами зазвичай повільніше за виконання результату компіляції. Це твердження дуже просто пояснити на прикладі наступного лістингу мовою C++:

```
1 #include <vector>
2 int main() {
3     std::vector<int> vec;
4     for (int i = 0; i < 100000; i++) {
5         vec.push_back(0);
6     }
7 }
```

Наведений програмний код відповідає програмі, що під час своєї роботи створює вектор цілих чисел після чого послідовно заповнює його нулями. У

разі построкового виконання програмного коду інтерпретатором, під час виконання рядка 3, буде виділений буфер пам'яті, розмір якого відповідатиме деякому значенню за замовчуванням. Після цього, виконання рядків 4-6 призведе до декількох послідовних перевиділень пам'яті кожен раз як розміру буфера буде не вистачати. У компілятора є доступ до рядків 4-6 під час вирішення якого розмір буфер слід виділяти, що означає, що необхідний обсяг пам'яті для 100000 елементів може бути виділений заздалегідь.

І хоча інтерпретатори відстають у продуктивності, користь їх гнучкості не слід недооцінювати. Можливість построкового виконання та активного втручання в роботу програми під цей час лягло в основу таких інтерактивних платформ для обчислень як Jupyter [85] та Matlab [71], перший з яких став одним з найпопулярніших середовищ для виконання наукових обчислень, а також обміну та представлення їх результатів [86].

Таким чином, оптимальним є метод трансляції розробленої мови, що поєднував би швидкість виконання обчислювально-складних задач з інтерактивністю у питанні їх задання та доступу до результатів.

3.2 Транслятор

Запропонована схема компіляції та використання, що задовольняє вищеназваним потребам, наведена у рисунку 3.1. Схема складається з наступних етапів:

1. Лістинг програми розробленою мовою опису моделі транслюється в проміжну мову. Цією проміжною мовою може бути Python або будь-яка мова програмування загального призначення, для якої існує компілятор та інструменти побудови зовнішніх модулів для мови Python;
2. У випадку, коли проміжна мова не є Python трансльований програмний код компілюється відповідним мові компілятором в імпортовану бібліотеку мови Python. Якщо проміжна мова є Python - результат трансляції і є імпортованою бібліотекою;

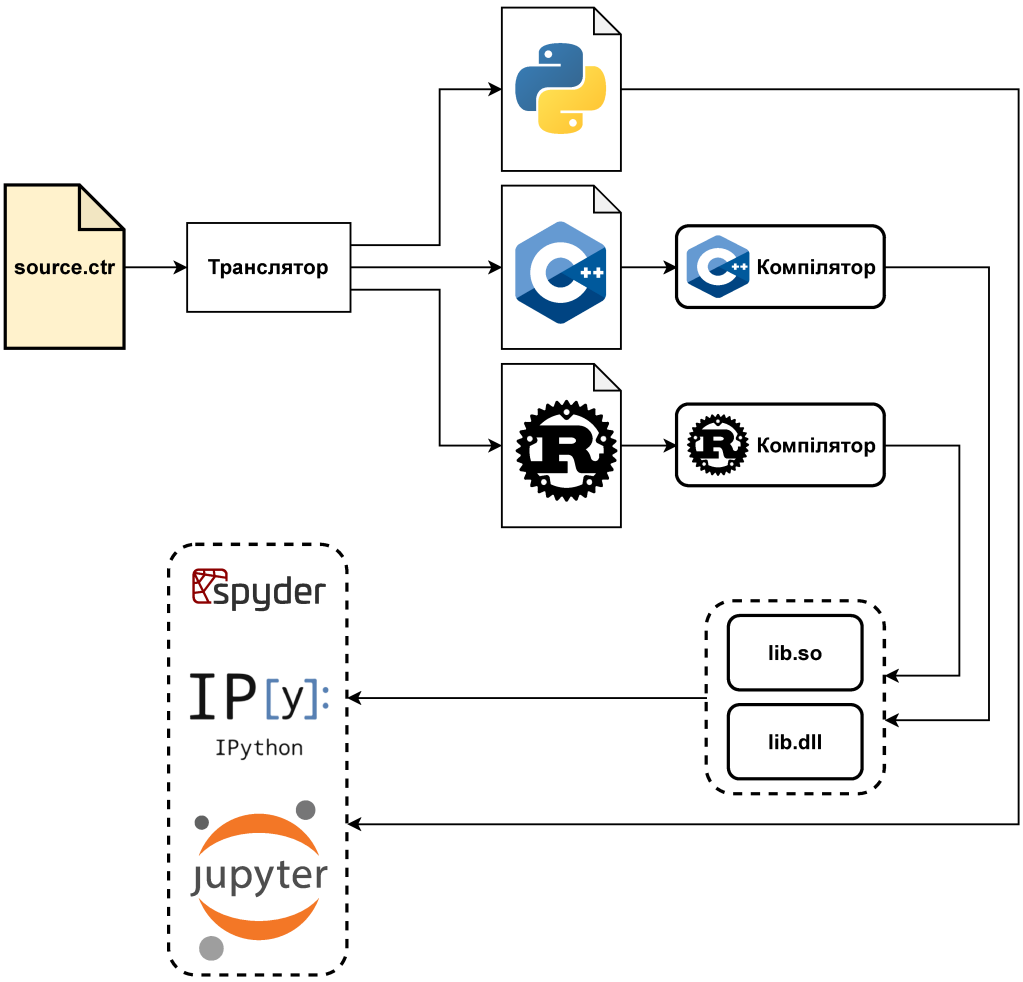

Рис. 3.1: Схематичне зображення процесу компіляції та використання розробленої мови

3. Згенерована бібліотека використовується у клієнтському програмному застосунку мовою Python.

І хоча мова Python не спроєктована під компіляцію, а також значно програє трансляторам таких мов як C++ або Rust у швидкодії результівної програми, використання бібліотек для JIT-компіляції таких як Numba [87], відповідно результатів з розділу 4, дозволяє звести цю різницю у швидкодії до мінімуму.

Сам процес трансляції у проміжну мову майже не залежить від цієї мови та складається з наступних етапів:

1. Лексичний аналіз, за якого вхідний лістинг програмного коду поділяється на рядок токенів;

2. Синтаксичний аналіз, за якого рядок токенів аналізується відповідно до правил формальної граматики мови та будується синтаксичне дерево лістингу програми;

3. Генерація вихідного коду проміжною мовою з синтаксичного дерева. Цей етап є різним для різних мов, адже повинен ураховувати структуру кожної.

Через те, що дослідження процесів лексичного та синтаксичного аналізів не є темою даної роботи, перші два етапи були реалізовані з використання бібліотеки Python Lex-Yacc (PLY) [88].

3.2.1 Оптимізації компілятора

Оптимізації компілятора, як було зазначено вище, дозволяють покращити код програми вихідною мовою у сенсі швидкодії, обсягу використаної пам'яті тощо. Оскільки головною проблемою у використанні глобальних епідеміологічних моделей є обсяг обчислень, усі оптимізації були спрямовані на мінімізацію часу обчислення одного кроку симуляції.

Через те, що мова не підтримує створення нових змінних під час роботи програми, обсяг пам'яті необхідний для її успішної роботи може бути обрахований на етапі компіляції. Це дозволяє звести до мінімуму кількість додаткових виділень пам'яті та збільшити швидкодію результівної програми. Наступний лістинг демонструє це на прикладі мінімальної програми, що описує поведінку 100 агентів у середовищі, де вони весь період симуляції перебувають в одному місці.

```
1 $compartment
2 Place[1] {
3 }
4 $agent
5 Person[100] {
6     schedule(h) = 0;
7 }
8 Flu {
```

```
    S 1, I 0;
    P(S -> I | I) = 0.1;
    P(S -> I) = 0.1;
}
```

У результаті компіляції генерується програмний код, частина якого наведена нижче:

```
M=np.zeros(shape=(234,), dtype=np.float64)
...
@njit
def _Person_schedule(M):
    pass
@njit
def Person_schedule(M,h,i):
    M[4]=h
    M[5]=i
    _Person_schedule(M)
    return 0.0
@njit
def Flu_state_prob_sample_multi(M, start_addr, n_elements):
    for i in range(n_elements):
        M[start_addr + i] = Flu_state_prob_sample(M)
@njit
def _Flu_progress_prob(M,out):
    s = M[212]
    d = M[213]
    out[:] = 0
    if s == 0:
        if True:
            out[1] = -0.15200309344504995
            return
@njit
def Flu_progress_prob(M,inf_state,inf_days,out):
    M[212]=inf_state
    M[213]=inf_days
    _Flu_progress_prob(M,out)
@njit
def _Flu_spread_prob_inc(M,out):
    sr = M[int(6.0 + M[218])]
```

```
33      sd = M[int(6.0 + M[219])]
34      if sd == 1:
35          if sr == 0:
36              out[1] += -0.15200309344504995
37          elif sr == 1:
38              pass
39  @njit
40  def Flu_spread_prob_inc(M, agent_r, agent_d, comp_i, out):
41      M[218]=agent_r
42      M[219]=agent_d
43      M[220]=comp_i
44      _Flu_spread_prob_inc(M, out)
45  @njit
46  def Flu_is_susceptible(M, r):
47      M[218]=r
48      return M[int(M[int(6.0 + M[218])] + 230.0)]
49  @njit
50  def Flu_is_infectious(M, d):
51      M[219]=d
52      return M[int(M[int(6.0 + M[219])] + 232.0)]
53  ...
```

Як можна побачити, усі функції використовують єдиний буфер пам'яті, що був оголошений у рядку 1.

Окрім того, для моделей, розмір яких менший за певне порогове значення, характерне для типу та моделі центрального процесора, що використовується для їх симуляції, стає можливим розміщення усього буферу в кеші процесора, що значно збільшує швидкодію.

Для константних виразів, тобто тих, визначення яких складається з констант та викликів вбудованих функцій, їх значення обраховується під час компіляції. Цей процес має назву згортка констант. Наприклад, наступне оголошення функції:

```
1  f(x) = x * (1 + 5);
```

транслюється в:

```
@njit
def _f(M):
    M[8]=M[7]*6.0

@njit
def f(M,x):
    M[7]=x
    _f(M)
    return M[8]
```

Як можна побачити, вираз $1+5$ був обрахований на етапі компіляції.

Алгоритми обрахунку $\mathcal{P}_\Delta(\mathbf{a},t)$ передбачають обчислення значної кількості добутків виду:

$$\underbrace{x_1x_1\ldots x_1}_{p_1}\ldots\underbrace{x_nx_n\ldots x_n}_{p_n} \tag{3.1}$$

Обчислення таких виразів може бути спрощене за допомогою наступної оптимізації:

$$\prod_{i=1}^{n}x_i^{p_i}=\exp\sum_{i=1}^{n}p_i\ln x_i \tag{3.2}$$

Якщо позначити за $t(*),t(+),t(\exp),t(\ln)$ час обрахунку добутку, суми, показникової функції та натурального алгоритму відповідно, час обчислення лівої частини буде рівним $t(*)(\sum p_i-1)$, правої - $(t(\ln)+t(+))\sum p_i-t(+)+t(\exp)$. У загальному випадку, останній вираз має більше значення, адже обчислення натурального логарифма є трудомісткою операцією. Проте, у випадку, коли значення x_i є константами, відомими на етапі компіляції, час обчислення правої частини буде дорівнювати $t(+)(\sum p_i-1)+t(\exp)$. Через те, що для обчислення добутку двох чисел з рухомою комою необхідна більша кількість циклів центрального процесора (для більшості архітектур) ніж для обчислення суми [89] обчислення оптимізованого варіанту є більш ефективним при:

$$t(*)-t(+)>\frac{t(\exp)}{\sum p_i-1}, \tag{3.3}$$

що є справедливим при достатньо великих значеннях $\sum p_i$. Таким чином, якщо функція розповсюдження захворювання залежить тільки від результівного стану та станів реципієнта та донора, зазначена оптимізація може бути використана. У рисунку 3.2 наведені результати порівняння швидкодії роботи моделі з та без використання зазначеної оптимізації.

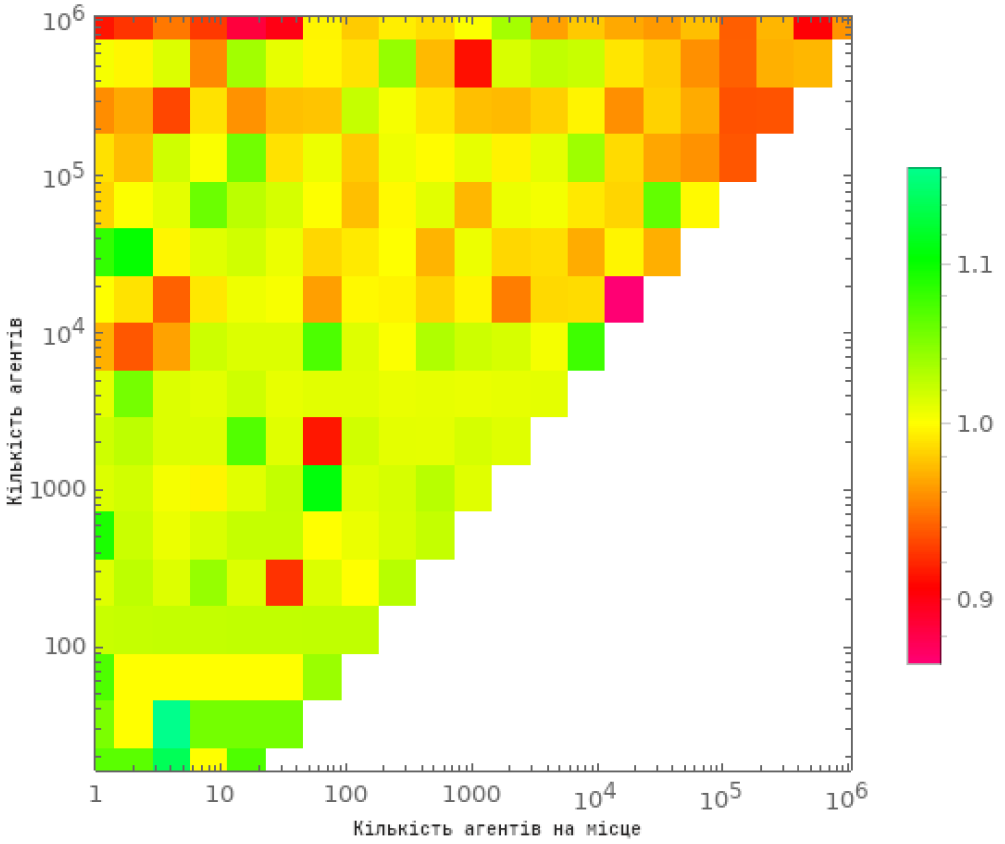

Рис. 3.2: Співвідношення часу обрахунку кроку моделі без оптимізації добутків та з нею для моделей з різною загальною кількістю агентів та кількістю агентів на місце (більші значення відповідають ефективній оптимізації).

3.3 Середовище розробки та аналізу

Розробка будь-якого програмного забезпечення тісно пов'язана з використанням інтерактивних середовищ розробки (англ. IDE). Ці середовища надають

такі функції як: підсвічування коду, підсвічування синтаксичних помилок, механізм автоматичного доповнення тексту, спрощений інтерфейс до транслятора тощо. Їх наявність значно спрощує та пришвидшує процес розробки [90]. Так, підсвічування коду дозволяє більш ефективно помічати ті чи інші синтаксичні структури у програмному коді, завдяки підсвічуванню помилок, розробник має змогу помічати синтаксичні помилки ще на етапі написання програмного коду, не викликаючи транслятор безпосередньо тощо.

Інтерактивним середовищем розробки моделей є середовище моделювання. Окрім вищеназваних функцій (що можуть бути присутні для моделей, заданих формальними мовами опису), таким середовищам характерна функціональність аналізу та візуалізації розроблених моделей. Подібно до синтаксичного аналізатора, візуалізація результатів моделювання паралельно процесу опису моделі дозволяє швидко помічати та виправляти помилки. Окрім цього, середовища моделювання можуть містити необхідний набір статистичних інструментів для виконання повного аналізу моделей.

З огляду на вищеназвані положення, було розроблене інтерактивне середовище розробки та аналізу агентних епідеміологічних моделей з використанням мови CTrace - CTraceEnv. Графічний інтерфейс користувача середовища CTraceEnv наведений у рисунку 3.3.

Інтерфейс складається з двох основних компонент: вікна опису моделі та її аналізу. Вікно опису моделі надає користувачу можливість описувати епідеміологічну модель за допомогою мови CTrace, використовуючи вбудований редактор програмного коду. Цей редактор містить у собі синтаксичний аналізатор, що у процесі редагування коду підсвічує наявні синтаксичні помилки. Окрім підсвічування коду, підсвічування синтаксису відображає окремі синтаксичні конструкції такі як константи, вбудовані ключові слова тощо окремими кольорами для більш наявного сприйняття. Відповідні кольори можуть бути задані у конфігураційному файлі.

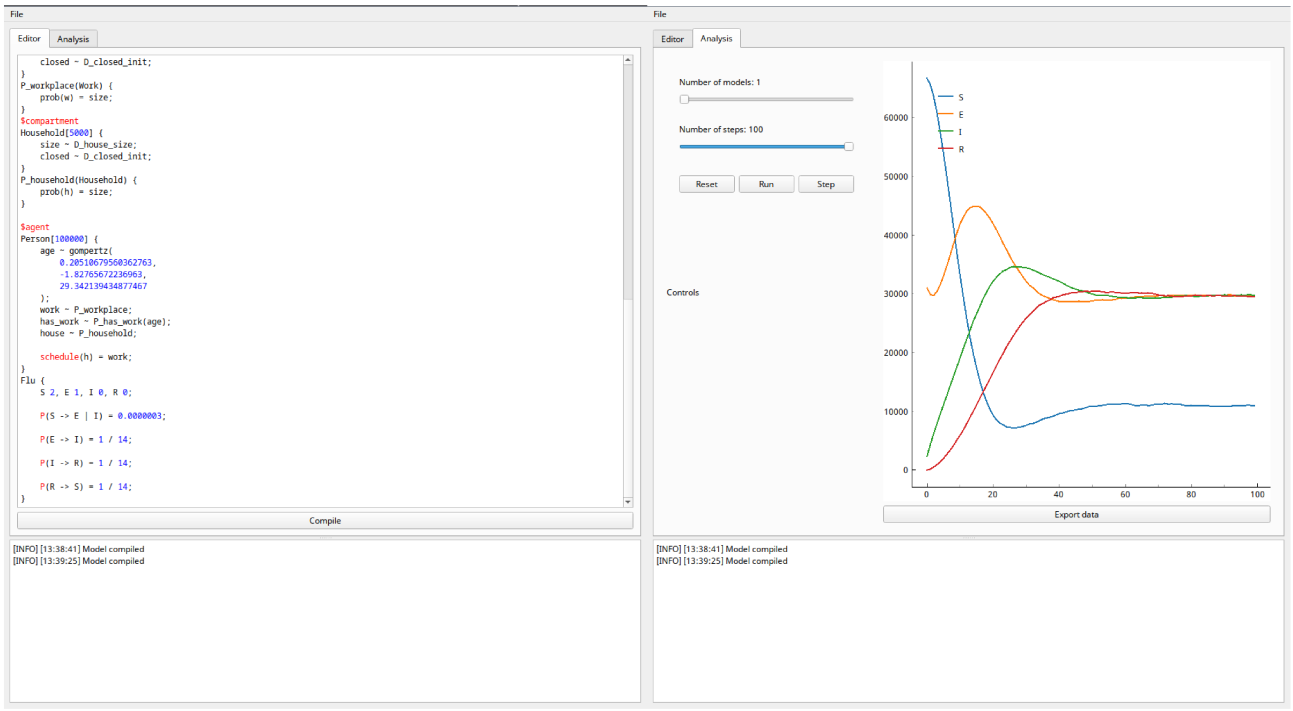

Рис. 3.3: Інтерфейс користувача середовища CTraceEnv. Вікно редактору коду – ліворуч, вікно аналізу моделі – праворуч

Вікно аналізу моделі надає функції базового аналізу заданої епідеміологічної моделі. Своєю чергою, ця компонента також має дві складові. Перша з них містить функціонал регулювання кількість кроків симуляції моделі та кількість моделей. Окрім цього, для моделей, що містять глобальні змінні (окрім кроку симуляції), інтерфейс містить відповідні їм елементи управління. Наприклад, для моделі, що у своєму описі містить задання параметрів температури навколишнього середовища та вологості, відповідний інтерфейс буде містити елементи для задання їх значень (рисунок 3.4).

Друга компонента містить графічне зображення динаміки кількості агентів, згрупованих за станом захворювання. У випадку, одночасного запуску декількох моделей, графічному зображенню підлягають середнє значення кількості агентів у кожній групі, а також 25-й та 75-й процентиль (рисунок 3.5).

Одним з важливих рішень у процесі проєктування середовища CTraceEnv є мінімальний інтерфейс аналізу. Замість того, щоб насичувати інтерфейс можливостями глибокого статистичного аналізу роботи моделей було вирішено

Рис. 3.4: Інтерфейс аналізу з елементами управління глобальними параметрами моделі

надати змогу експортувати сирі дані їх роботи для подальшого аналізу з використанням спеціалізованих програмних інструментів.

Як вікно редагування моделі, так і її аналізу мають спільну компоненту – вікно логування, що відображає стан роботи програми, у тому числі наявні помилки компіляції. Середовище CTraceEnv виконує безперервне збереження поточного коду опису моделі та дає змогу відновити його у випадку аварійного завершення роботи. Також, присутній базовий функціонал збереження/завантаження описів моделі із файлової системи.

Для забезпечення підтримки крос-платформеності, середовище CTraceEnv було імплементоване з використаннями мови програмування Python та бібліотек програмування графічного інтерфейсу користувача PyQtGraph [91] та PyQt [92]. Підтримка мови CTrace виконана за допомогою розроблених інструментів трансляції, описаних у розділі 3.2.

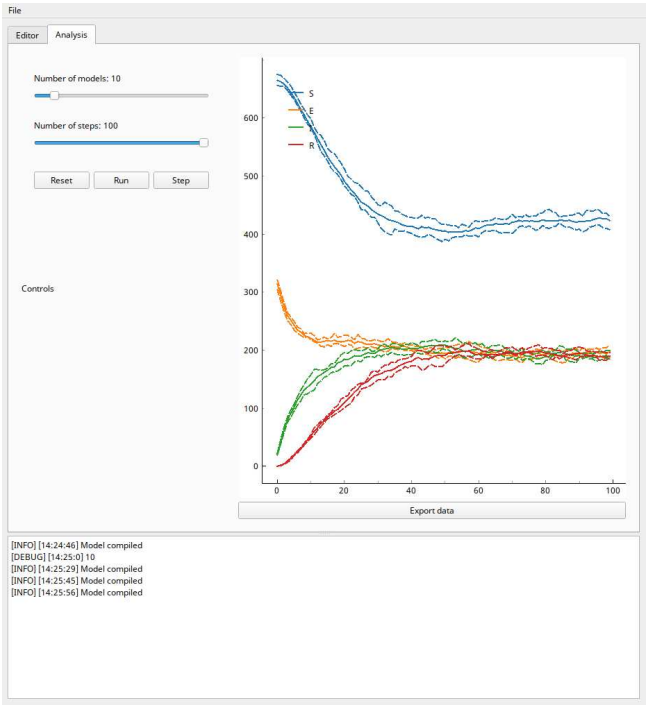

Рис. 3.5: Графічне зображення епідеміологічної динаміки декількох моделей. 25-й та 75-й процентилі позначені пунктиром

ВИСНОВКИ ДО ТРЕТЬОГО РОЗДІЛУ

У рамках третього розділу була виконана побудова транслятора розробленої формальної мови опису агентної епідеміологічної моделі, а також відповідного середовища моделювання.

Розроблений транслятор є компілятором із конфігурованою проміжною мовою. Такою мовою може бути Python, який своєю чергою транслюється у мову машинних інструкцій під час процесу JIT-компіляції, або будь-яка мова загального призначення, що має інтерфейс до побудови Python-модулів. Такою мовою була обрана мова Rust. Основним аспектом алгоритму трансляції є мінімізація кількості виділень пам'яті у процесі виконання результівної програми. Для цього, компілятор обраховує загальний обсяг пам'яті, необхідний для роботи моделі, і використовує буфер відповідного розміру для організації усіх обчислень.

Розроблене середовище розробки та аналізу агентних епідеміологічних моделей з використанням мови CTrace – CTraceEnv. Розроблене середовище надає базовий функціонал необхідний для підтримки процесу опису моделей мовою CTrace, що включає підсвітку синтаксису вихідного коду, вбудований транслятор тощо. Серед функціоналу аналізу представлені елементи керування роботою моделі, значень її глобальних параметрів, інтерфейс перегляду динаміки розповсюдження досліджуваного інфекційного захворювання, а також функціонал експорту результатів для подальшого аналізу сторонніми спеціалізованими інструментами.

РОЗДІЛ 4.

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ МОВИ CTRACE ТА СЕРЕДОВИЩА CTRACEENV У ПРОЦЕСІ РОЗРОБКИ АГЕНТНИХ МОДЕЛЕЙ РОЗПОВСЮДЖЕННЯ ІНФЕКЦІЙНИХ ЗАХВОРЮВАНЬ

4.1 Аналіз впливу соціодеографічної гетерогенності на стратегії впровадження карантинних заходів

Соціодемографічна гетерогенність є мірою різноманітності стану сутностей середовища епідеміологічного моделювання, що може бути виражене у різноманітності ймовірнісних розподілів віку, статі, рівнів доходу населення тощо. Урахування цієї різноманітності є доволі складним аспектом у процесі імплементації моделей. Тому постає питання: чи є воно обумовленим більш ефективними стратегіями впровадження епідеміологічних заходів, побудованих на основі моделей з детальною параметризацією. Для перевірки був проведений експеримент, що складається з наступних кроків:

1. Побудувати набір агентних епідеміологічних моделей, параметризованих соціодемографічними даними різних країн;
2. Для кожної моделі визначити субоптимальну стратегію впровадження карантинних заходів для неї;
3. Провести перехресне порівняння стратегій на моделях відповідних їм країн та країн, відповідним іншим стратегіям.

4.1.1 Тестова епідеміологічна модель

Тестова епідеміологічна модель є представлена множиною агентів P та множинами місць їх перебування H, S, W, що відповідають приватним будинкам, закладам освіти та роботи. Кожен агент параметризований його віком, зайнятістю, рівнем освіти та станом захворювання, що під час ініціалізації моделі за-

даються відповідними ймовірнісними розподілами. На основі цих параметрів, а також на основі рівня карантинних обмежень, формується функція розкладу, що диктує переміщення агентів у рамках середовища. Перебіг захворювання моделюється з використання станів, відповідним моделі SEIR. Дискретність симуляції – одна астрономічна година.

Зазначені вище розподіли були задані на основі відкритих демографічних даних наданих Statistical Office of the European Communities (Eurostat). Зокрема, були використані наступні джерела даних:

- Families by type, size and NUTS 3 region;
- Population on 1 January by age and sex;
- Pupils and students enrolled by education level, sex and age;
- Employment by sex, age and NUTS 2 regions (1 000).

4.1.2 Визначення субоптимальної стратегії впровадження карантинних заходів

Для визначення субоптимальної стратегії впровадження карантинних заходів, був використаний алгоритм Deep Q-learning with experience replay [93], що є представником алгоритмів навчання з підкріпленням [94]. Цей алгоритм знаходить стратегію, що максимізує майбутню винагороду дій агента у межах середовища. Агент, його дії, винагорода та середовище мають бути сформульовані як Марковський процес прийняття рішень [95].

Діями агенту у процесі пошуку стратегій впровадження карантинних заходів є безпосередньо їх впровадження або скасування. Так, у рамках тестових моделей була запропонована наступна множина дій: $A = \{0, 1, 2\}$, де 1 - впровадження повного карантину, що передбачає карантин усіх закладів; 2 - скасування карантину; 0 - відсутність будь-яких дій. Станом середовища є сукупність інфекційних станів кожного агента, а також карантинних станів кожного закладу.

Винагорода за дію агента складається з двох компонент: економічної та епідеміологічної. Перша відображає вплив карантинних заходів на економіку і контролює їх агресивність, запобігаючи нанесення занадто сильної шкоди. Друга компонента безпосередньо показує ефективність дій агента у процесі контролю за розповсюдженням інфекційного захворювання. Таке розбиття є необхідним, адже за відсутності економічної складової оптимальна стратегія, що максимізує епідеміологічну винагороду є $\pi(s) = 2, \forall s \in S$, що не є допустимим у реальному світі.

Агент являє собою повнозв'язну штучну нейронну мережу з одним шаром і функцією активації ReLU [96]. Оптимізація ваг нейронної мережі виконувалось із використанням алгоритму Adam [97]. Детальний опис наведений у [17].

Було виконане порівняння знайдених стратегій із тривіальними, до яких належать:

- $\pi(s) = 1$ - повний карантин;
- $\pi(s) = 2$ - повна відсутність карантину;
- $\pi(s) \sim \mathcal{U}\{0, 2\}$ - випадкові дії.

Результати порівняння наведені у рисунку 4.1. З результатів очевидно, що знайдені стратегії є більш ефективними ніж тривіальні.

4.1.3 Перехресне порівняння стратегій

Перехресне порівняння стратегій полягає у порівнянні сумарної винагороди за дії агента, що був натренований на одній моделі однієї країни у середовищі інших країн. Тобто, показником ефективності роботи агента на моделях інших країн є значення:

$$E_{c_1,c_2} = \frac{\sum\limits_t R_t^{(c_2)}}{\sum\limits_t R_t^{(c_1)}}, \tag{4.1}$$

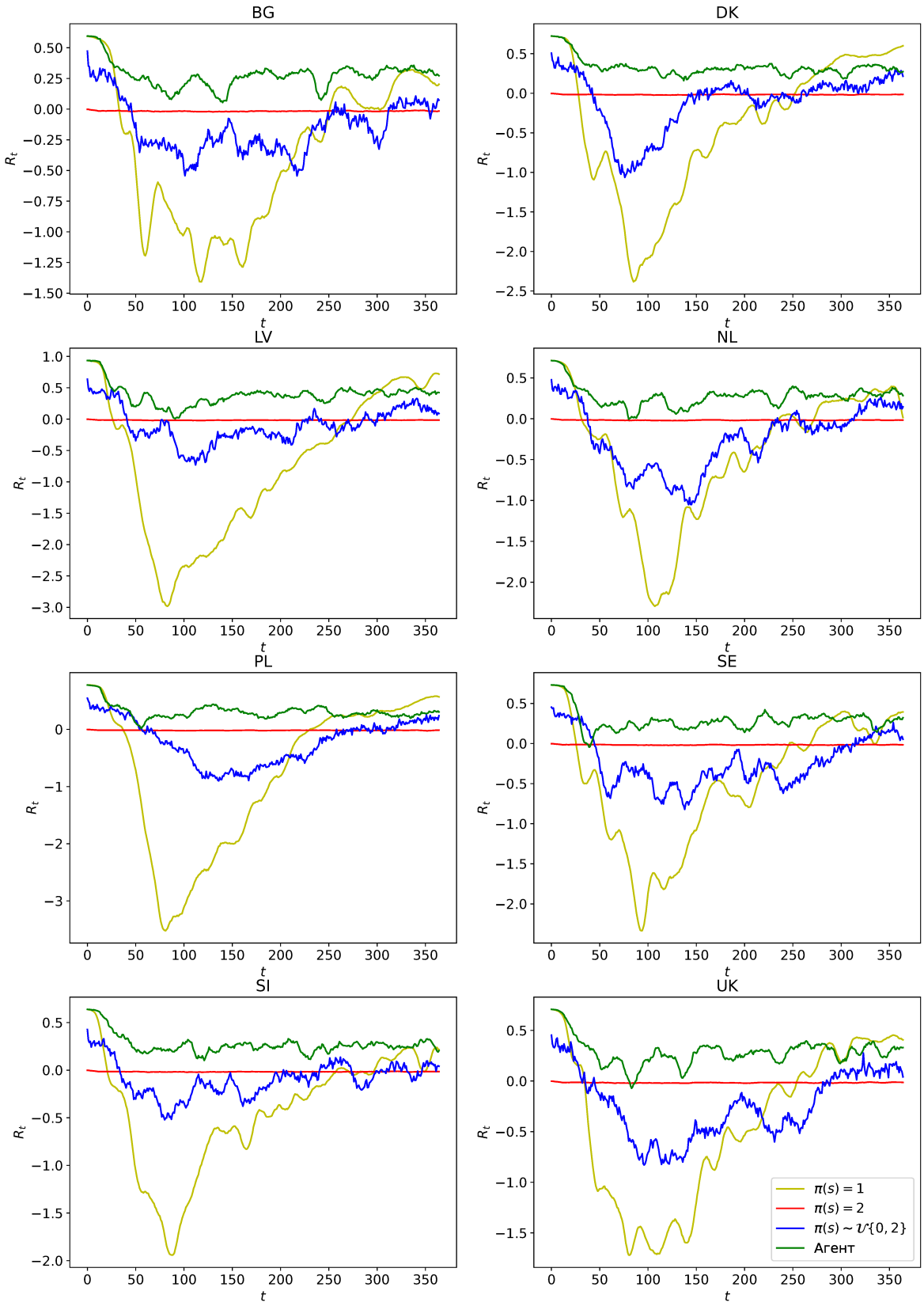

Рис. 4.1: Винагорода за дії агента протягом часу симуляції для різних стратегій запровадження карантинних заходів. Зелена крива відповідає знайденій субоптимальній стратегії. Зверху кожного графіку зазначена країна тестування

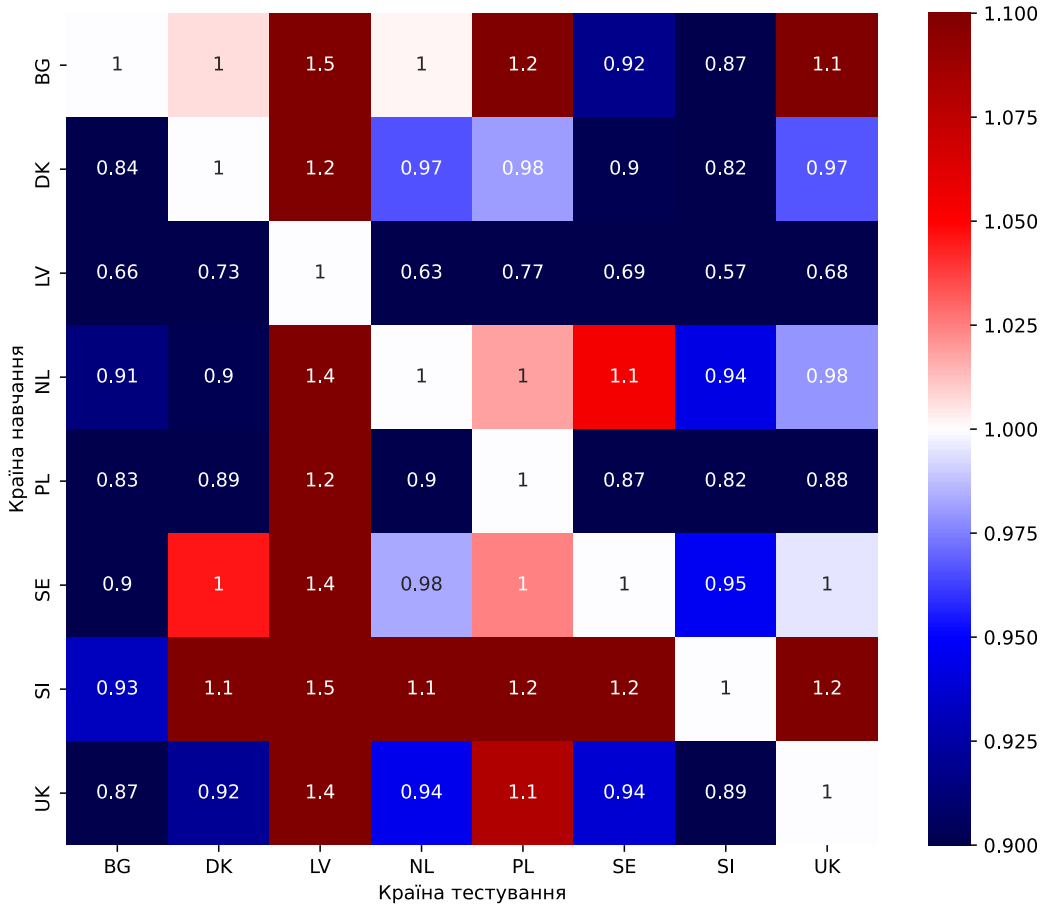

Рис. 4.2: Значення E_{c_1,c_2} для різних країн навчання агенту та тестування

де c_1 - країна тренування агента, c_2 - країна тестування агента, $R_t^{(c)}$ - винагорода у час t для середовища країни c.

У рисунку 4.2 зазначені значення E_{c_1,c_2} для усіх пар обраних країн.

Виходячи з того, що для 62.5% пар країн, агент, натренований на моделі країни тестування є більш ефективними ніж інші агенти, а також з факту того що єдина відмінність між моделями країн – різниця їх соціодеографічних параметрів, можна зробити висновок про значний цих параметрів на фінальну стратегію впровадження карантинних заходів.

4.2 Моделювання розповсюдження коронавірусу SARS-CoV-2 з використанням мови CTrace

Для оцінки придатності розробленої загальної агентної епідеміологічної моделі, а також мови CTrace для епідеміологічного моделювання була побудована та відкалібрована тестова епідеміологічна модель розповсюдження коронавірусу SARS-CoV-2 серед населення Польщі у період з початку вересня 2020 року до кінця листопада 2020 року. Структура моделі аналогічна використаній у минулому підрозділі. Джерело соціодемографічних даних – бази надані Eurostat для Польщі. Джерело епідеміологічних даних – Worldometer [98]. Процес калібрування полягав у вирішенні задачі мінімізації відхилення модельованої динаміки розповсюдження від істинної, що може буди задане наступним виразом:

$$\underset{\Phi_t \times \Phi_p}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (I_{t+i} - \sum_{a \in \mathbf{A}, \xi(a,t+i) = ``I''} 1)^2, \tag{4.2}$$

де I_i – кількість інфікованих людей у час i, відповідно до історичних даних. Мінімізація була виконана з використанням алгоритму Tree-structured Parzen Estimator [99].

Детальний опис результівної моделі наведений у додатку В.

З рисунка 4.3 видно, що розроблена загальна епідеміологічна модель здатна моделювати реальні епідеміологічні дані, що свідчить про доцільність її використання у даній предметній області.

4.3 Аналіз семантики розробленої мови опису моделей

Важливим аспектом мови опису моделей є її гнучкість, тобто обсяг множини моделей та їх типів, що може бути описана з її використанням. Очевидним є факт, що доцільність використання мови, здатної описувати один конкретний тип моделі є доволі низькою, адже у випадку зміни напряму розвитку дослі-

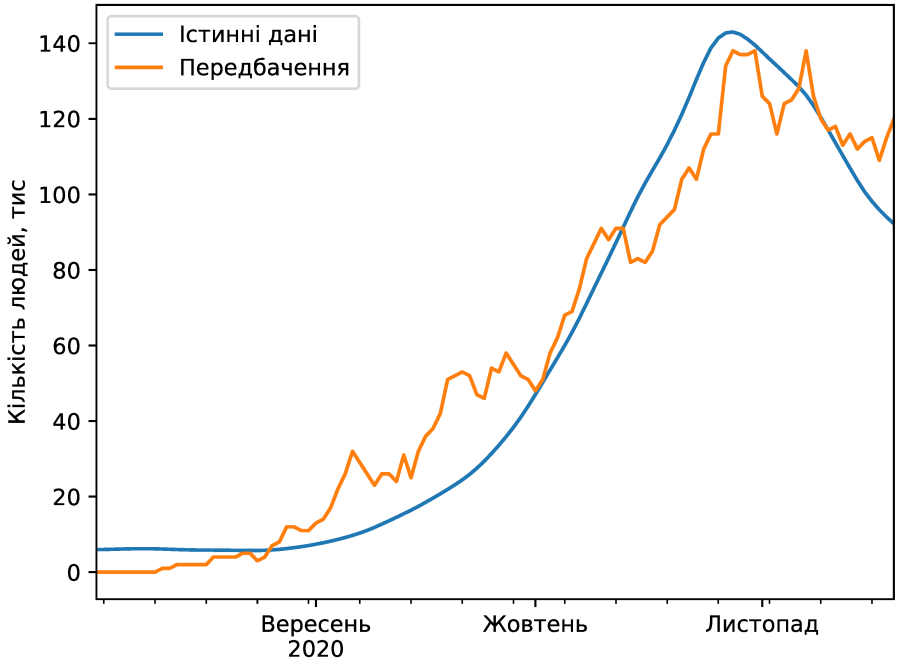

Рис. 4.3: Кількість інфікованих людей серед населення Польщі відповідно до реальних даних та згідно з розробленою моделлю. $R^2 = 0.9319$

джень у предметній області, дана мова потребує модифікування. Гнучкість мови не є аналітичною величиною, тому для якісної її оцінки для розробленої мови моделювання були виокремлені основні аспекти та принципи побудови сучасних епідеміологічних моделей та було показано як їх відтворити мовою CTrace.

4.3.1 Динамічні параметри моделі та її сутностей

Задання параметрів моделі, агентів та інших сутностей як динамічних величин, значення яких може змінюватись відповідно до створених розробником правилам є необхідним аспектом у побудові реальних епідеміологічних моделей. Наприклад, такий параметр як вік у певних випадках можна прийняти константним (у разі короткострокового моделювання). В інших – урахування його зміни є необхідним. Мова CTrace дозволяє задавати такі параметри разом з правилами

їх змін. У лістингу нижче наведений приклад такого задання.

```
t = 0;
t -> t + 1;
$agent
Human[100] {
    age ~ D_age;

    age -> age + 1;
}
```

Даний приклад розглядає модель, що має два динамічних параметри: t та *age*, що збільшуються на один кожну ітерацію моделі. Параметр *age* характерний кожному агенту. Мова дозволяє описувати й складніші правила зміни значень параметрів за допомогою використання функцій та ймовірнісних розподілів.

4.3.2 Моделювання географії середовища

У процесі епідеміологічного моделювання неможливо виключати аспект географічного розташування певних сутностей моделі. Так, наприклад фізичне віддалення населених пунктів може слугувати своєрідним бар'єром у процесі розповсюдження захворювання. Для урахування даної особливості моделей, сутності, для яких притаманне географічне розташування можуть містити географічні координати як параметри. У лістингу нижче, наведений фрагмент опису епідеміологічної моделі з урахуванням географії.

```
D_workplace_lat {
    ...
}
D_workplace_lng(lat) {
    ...
}

$place
Workplace[10] {
    lat ~ D_workplace_lat;
```

```
    lng ~ D_workplace_lng(lat);
}
```

Сутність Workplace, містить свої географічні координати – широту та довготу. Ці параметри задаються ймовірнісним розподілом D_workplace_lat, що задає широту та умовним розподілом D_workplace_lng. Ці координати можуть буди використані у подальшому заданні, наприклад, функції розповсюдження.

Окремим підходом до моделювання є розташування сутностей моделі в комірках сітки. У такому випадку, інфікування відбувається лише для агентів у сусідніх комірках. Фрагмент опису моделі такого типу наведений у лістингу нижче.

```
D_agent_x {
    ...
}
D_agent_y(agen_x) {
    ...
}

$compartment
Space[1] {}
P_space(Space) {
    prob(w) = 1;
}

$agent
Agent[10] {
    x ~ D_agent_x;
    y ~ D_agent_y(x);
    place ~ P_space;
}

Disease {
    S 0.9, I 0.1, R
    P(S -> I | I) = check_if_adjacent(a, d) * p;
}
```

Аналогічно минулому прикладу, сутність Agent містить свої географічні ко-

ординати. Функція розповсюдження захворювання Disease використовує функцію check_if_adjacent, що здійснює перевірку агентів на сусіднє розташування. Таким чином, якщо ця функція повертає 0 у випадку віддаленості агентів, інфікування одного з них іншим стає неможливим.

4.3.3 Моделювання векторних захворювань

Векторні захворювання – тип захворювань, у процесі розповсюдження грають роль вектори. Вектор, або переносник – комаха, або інший живий організм, що транспортує інфекційний агент від інфікованого організму до сприйнятливого [100]. Для опису моделей таких захворювань мовою CTrace кожен агент може містити параметр, що вказує чи є він вектором, чи ні. Також, через те, що векторам характерний окремий тип перебігу захворювання, стан інфікування має бути модифікований відповідно. Приклад опису моделі векторного захворювання наведений у лістингу нижче.

```
1  D_vector {
2      ...
3  }
4
5  $agent
6  Agent[1000] {
7      is_vector ~ D_vector;
8  }
9
10 Disease {
11     In 1, S 0, I 0.1, R 0, Sv 0;
12
13     P(In -> S) = 1 - d.is_vector;
14     P(In -> Sv) = d.is_vector;
15 }
```

У наведеному лістингу розподіл D_vector вказує на початковий розподіл векторів у модельованій популяції.

4.3.4 Урахування погодних умов

Погодні умови такі як температура, вологість повітря тощо має вплив на динаміку розповсюдження інфекційного захворювання [101]. Для задання цих параметрів в описі моделі мовою CTrace достатньо оголосити відповідні глобальні змінні та, за необхідності, правила їх зміни. Ці змінні надалі можуть бути використані у заданні інших аспектів моделі, наприклад функції розповсюдження. Фрагмент коду моделі, що включає задання температури та вологості наведений у лістингу нижче.

```
1 TEMPERATURE_C = 22;
2 get_humidity_p() = some_function(T);
```

У наведеному лістингу, TEMPERATURE_C - глобальна змінна, що задає температуру повітря у градусах Цельсія; get_humidity_p - функція, що за певним правилом обраховує вологість повітря на основі кроку симуляції (часу).

Окрім зазначених аспектів було виконане загальне порівняння функціонала для епідеміологічного моделювання у наявних методах та інструментах з доступним у середовищі CTraceEnv. Результати наведені у таблиці 4.1. Для виконання порівняння, був введений показник функціонала, високі значення якого показують значний загальний обсяг доступних для моделювання сценаріїв розповсюдження інфекційних захворювань. Значення показника дорівнює кількості характерних методу ознак з таблиці 4.1.

Таблиця 4.1: Порівняння доступного функціоналу у наявних методах/інструментах епідеміологічного моделювання

	CTraceEnv	FluTE	FRED	NetLogo	GiacopelliLombardy [102]	Kendrick	GLEaM	STEM
Компартментні моделі	-	-	-	-	-	+	+	+
Агентні моделі	+	+	+	+	+	-	-	-
Задання користувацьких станів захворювання/компартментів	+	-	-	+	-	+	+	-
Урахування атрибутів окремих представників популяції	+	+	+	+	+	-	-	-
Задання атрибутів представників популяції	+	+	-	+	-	+	-	-
Динамічні значення атрибутів представників популяції	+	-	-	+	-	+	+	+
Нефіксована просторова роздільна здатність	+	-	-	+	-	-	-	+
Нефіксована часова роздільна здатність	+	-	-	+	-	-	-	-
Вбудована Інтеграція з джерелами статистичних даних	-	-	-	-	-	-	-	+
Паралельне/розподілене обчислення	-	+	+	-	+	-	+	+
Графічний інтерфейс користувача	+	-	+	+	-	+	+	+
Динамічні параметри моделі	+	-	-	+	-	-	+	-
Нефіксовані типи місць	+	-	-	+	-	-	-	-
Векторні захворювання	+	-	-	+	-	+	+	+
Моделювання вакцинації	+	-	+	+	+	+	+	+
Моделювання сезонних явищ	+	-	+	+	-	+	+	+
Моделювання географії	+	+	+	+	+	-	+	+
Урахування погодних умов/клімату	+	-	-	+	-	+	-	-
Показник функціональності	**15**	**5**	**7**	**15**	**5**	**9**	**10**	**10**

4.4 Аналіз ефективності розробки епідеміологічних моделей мовою CTrace у середовищі CTraceEnv

Важливим аспектом оцінки будь-якої формальної мови є час її використання для розв'язання певної задачі. Наприклад, мова асемблера хоча і є надшвидкою у сенсі ефективності виконання програмних інструкцій, не підходить для опису та програмування епідеміологічних моделей через значний час необхідний для написання програм з її допомогою. Через те, що неможливо точно оцінити реальну кількість людино-годин, витрачених на розробку наявних епідеміологічних моделей (через те, що вони вже розроблені й такі виміри потрібно робити до початку їх розробки), було вирішено оцінити ефективність розробки епідеміологічних моделей мовою CTrace як співвідношення об'єму коду проміжною мовою трансляції та вихідного коду опису моделі. Цей об'єм був заданий як кількість символів у файлах вихідного коду окрім символів, що належать коментарям.

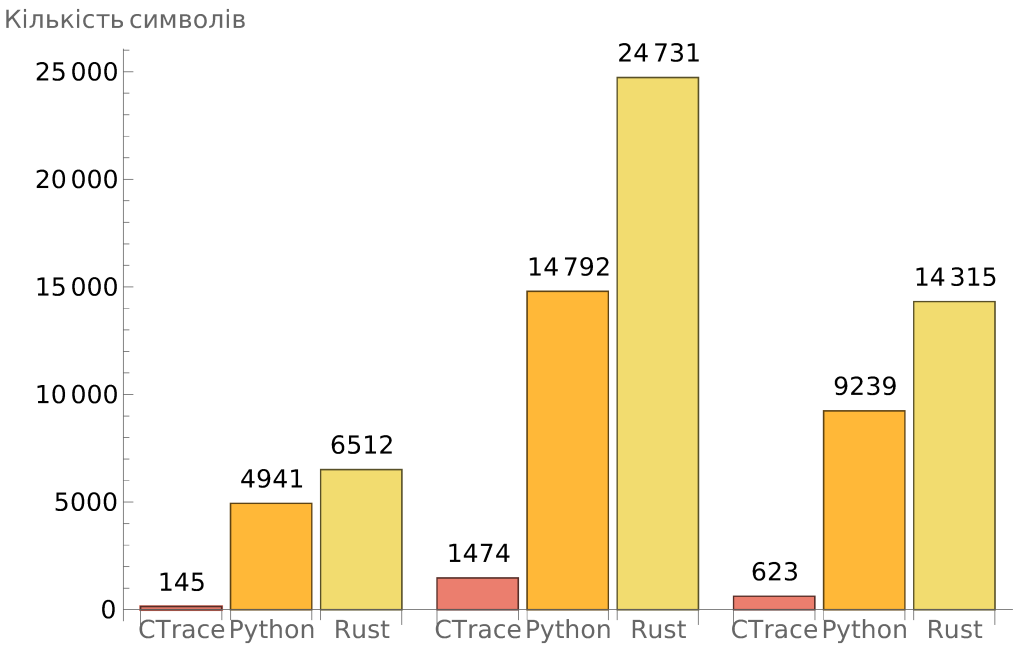

Рис. 4.4: Порівняння об'єму вихідного коду опису моделі та коду проміжною мовою трансляції. Зліва направо: Тривіальна модель, модель побудована на основі [56], модель побудована на основі [51]

Виходячи з отриманих експериментальних даних можна зробити висновок, щодо ефективності розробки агентних епідеміологічних моделей мовою CTrace. Вона дозволяє значно скоротити обсяг кодової бази відповідних проєктів (для мови Python кодова база зменшилась у 14–34 рази, для мови Rust – у 16-45 разів), що може свідчити й про скорочення часу їх розробки. І хоча відтворені моделі не є ідеальною копією оригіналів, основні механізми їх роботи були імплементовані. У Додатку А наведені лістинги опису моделі побудованої на основі [56] мовою CTrace, а також проміжним мовами.

4.4.1 Аналіз швидкодії проміжної мови трансляції

З огляду на те, що мова Python не відрізняється значною швидкодією, був проведений експеримент, завдання якого було показати доцільність її використання як проміжної мови трансляції. У рамках експерименту була побудована тестова модель, швидкодія якої була заміряна з використанням проміжних мов Python та Rust при різних її розмірах. Для мови Python були виконані окремі заміри з та без використання бібліотеки JIT-компіляції Numba. Результати експерименту наведені у рисунку 4.5 та таблиці 4.2. Можна побачити, що використання бібліотеки Numba для JIT-компіляції значно зменшує час обчислення кроку симуляції та робить його приблизно рівним такому для мови Rust. З цих результатів можна зробити висновок, що використання мови Python як проміжної у процесі трансляції є доцільним.

4.4.2 Аналіз швидкодії епідеміологічних моделей, описаних мовою CTrace

Для виконання процесу аналізу швидкодії моделей заданих мовою CTrace, було виконано їх порівняння з наявними моделями. Для цього, була розроблена епідеміологічна модель з типовими механізмом роботи, характерним іншим моделям (розклад дій агентів, різні типи компартментів тощо). Експериментальним шляхом була оцінена її швидкодія та виконане порівняння з іншими підходами.

Таблиця 4.2: Час обчислення одного кроку симуляції тестової епідеміологічної моделі для мови Python з та без використання JIT-компіляції. Значення нормовані відносно значень для мови Rust

Кількість агентів	Python				
	Мін.	25-й процентиль	Середнє	75-й процентиль	Макс.
100	419.035	411.943	399.686	380.577	363.047
300	408.706	412.517	385.383	382.479	245.906
1000	410.920	416.043	386.725	368.656	364.410
3000	404.897	406.450	403.540	420.085	321.436
10000	409.037	414.989	420.488	435.577	311.462
30000	369.004	363.273	350.933	332.575	332.566
100000	282.011	277.874	273.090	274.690	217.800
300000	243.724	240.710	243.500	245.815	236.803
1000000	182.765	191.264	188.286	188.262	182.382
Кількість агентів	Python+Numba				
	Мін.	25-й процентиль	Середнє	75-й процентиль	Макс.
100	1.332	1.405	1.303	1.280	1.030
300	1.324	1.347	1.249	1.231	0.991
1000	1.324	1.331	1.196	1.143	1.067
3000	1.320	1.336	1.336	1.375	1.140
10000	1.366	1.347	1.421	1.485	1.323
30000	1.336	1.335	1.351	1.312	1.421
100000	1.188	1.165	1.173	1.184	1.131
300000	1.166	1.174	1.193	1.203	1.334
1000000	1.012	1.004	1.033	1.017	1.193

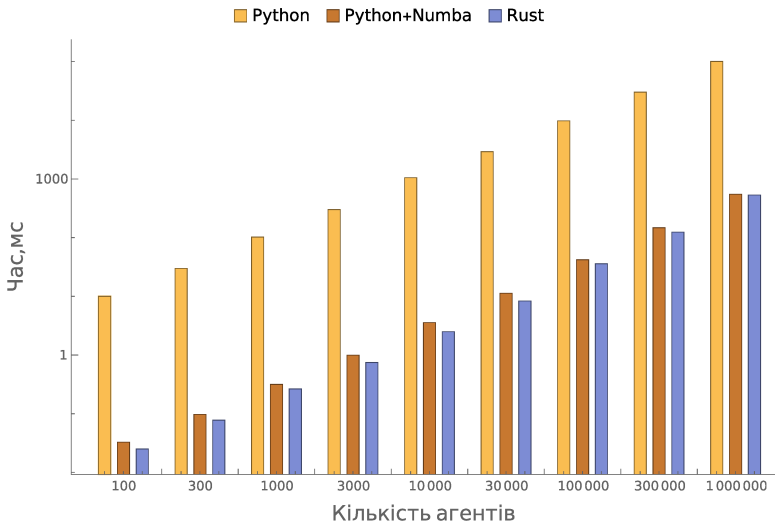

Рис. 4.5: Час обчислення одного кроку симуляції тестової епідеміологічної моделі для різних проміжних мов трансляції

Через те, що час обрахунку агентної моделі залежить від кількості симульованих днів, кількості кроків у дні, кількості агентів та кількості процесорних ядер, долучених паралельно до обчислення, була запропонована метрика, що показує кількість ЦП-секунд, необхідна для обрахунку одного кроку моделі, що містить один мільйон агентів та обраховується за формулою:

$$\mathbb{T} = \frac{T}{10^{-6} N_a * N_d * N_s * N_c}, \tag{4.3}$$

де, T - час обрахунку моделі у секундах, N_a — кількість агентів, N_d — кількість днів, N_s — кількість кроків, N_c — кількість процесорів.

Виходячи з наведених експериментальних даних можна зробити висновок, що швидкодія моделей, описаних мовою CTrace з використанням розробленого транслятора не уступає швидкодії інших моделей.

У рисунку 4.6 наведена залежність показника функціональності від $\mathbb{T}$. Можна помітити, що розроблене середовище CTraceEnv виграє у всіх зазначених підходах та інструментах у показнику функціональності окрім NetLogo, що є очікуваним, адже NetLogo є середовищем загального моделювання. Проте, NetLogo значно програє у швидкодії.

Таблиця 4.3: Значення метрики ефективності для розглянутих методів та інструментів агентного моделювання розповсюдження інфекційних захворювань

Метод	$\mathbb{T}$, c	Обладнання
FluTe[74]	4.00 — 13.71	32x Intel Core2 Duo T9400
AvilovNetLogo[51]	511.36 — 10960.15	Intel Core i3 330M 2.13Ghz
AvilovOracle[51]	5.75 — 13.71	
FRED[77]	1.20 — 7.29	SGI Altix UV supercomputer
MontañolaNetLogo[103]	21.57	Intel Core i5 3.20 GHz
GiacopelliLombardy[102]	240	AMD 3900X 12-core
CTrace	**0.67**	**Intel Core i7-8700K 3.70 GHz**
	1.38	**Intel Core i3-2330M 2.20 GHz**

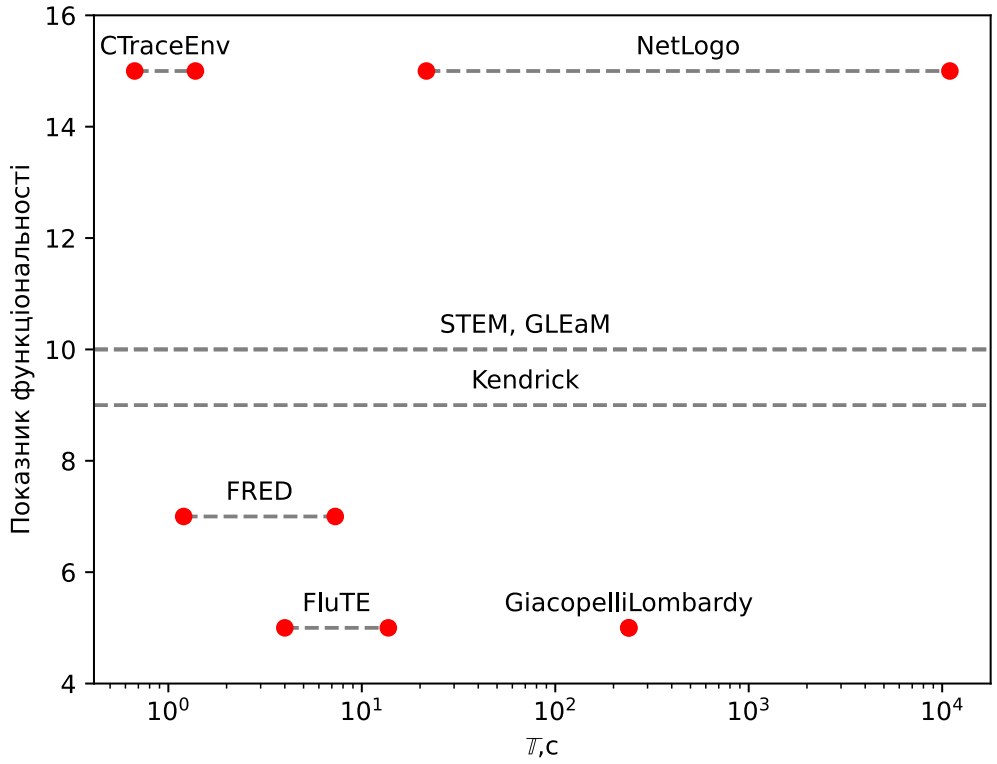

Рис. 4.6: Залежність показника функціональності від $\mathbb{T}$. Червоними крапками позначені мінімальні та максимальні значення $\mathbb{T}$

ВИСНОВКИ ДО ЧЕТВЕРТОГО РОЗДІЛУ

У рамках цього розділу був проведений аналіз розробленої мови опису агентних епідеміологічних моделей з боку доцільності її використання, обсягу сценаріїв, що можуть бути змодельовані, ефективності розробки та використання.

Виконана практична апробація розробленого інструментарію для ефективного моделювання розповсюдження інфекційних захворювань. У її рамках, було дано визначення агентної епідеміологічної моделі розповсюдження коронавірусу SARS-CoV-2 серед населення Польщі у період з початку вересня 2020 року до кінця листопада 2020 року, що була відкалібрована з використанням публічно доступних епідеміологічних даних. Результівна модель описує динаміку зміни кількості інфікованих людей з коефіцієнтом детермінації рівним 0.9319, що свідчить про здатність розробленого підходу описувати процеси розповсюдження інфекційних захворювань.

Проведений аналіз впливу соціодемографічної гетерогенності середовища моделювання на якість карантинних стратегій побудованих на його основі. Відповідний експеримент полягав у реалізації алгоритму пошуку карантинних стратегій для заданої агентної епідеміологічної моделі країни Європейського Союзу. Соціодемографічна параметризація моделі побудована з використанням статистичного моделювання на основі відкритих історичних даних, з відкритої бази статистичних даних Eurostat. Знайдені алгоритмом стратегії були порівняні із тривіальними, що показало більшу їх ефективність та використані у процесі перехресного порівняння. Для цього, стратегія, отримана з моделі країни тестування була порівняна з іншими. В обох випадках критерієм порівняння виступала сумарна винагорода за дії агента, що використовує відповідну стратегію у середовищі моделювання, сформульованим як Марківський процес вирішення. Аналіз показав, що для покращення їх якості соціодемографічні показники мають бути враховані, що, своєю чергою, показує необхідність наявності їх задання у будь-якому інструменті епідеміологічного моделювання. Це

свідчить про доцільність тої уваги, що була приділена процесу їх задання при розробці мови CTrace.

Окремо був проведений якісний аналіз спектра сценаріїв, що можуть бути змодельовані з використанням мови CTrace. Показано моделювання таких аспектів агентного епідеміологічного моделювання як: динамічні параметри моделі та її сутностей, географія середовища, погодні умови, векторні захворювання.

Аналіз ефективності розробки епідеміологічних моделей мовою CTrace показав значне скорочення обсягу вихідного коду програми у порівнянні з моделями імплементованими мовами програмування загального призначення. В якості цих моделей, був використаний лістинг програмного коду проміжною мовою, що генерується транслятором. Скорочення обсягу програмного коду становило від 14 до 34 разів для мови Python та від 16 до 45 разів для мови Rust. Це скорочення може свідчити про зменшення кількості людино-годин необхідних для їх розробки, а також збільшену швидкість ітерування. Окрім того, що зі зменшенням обсягу вихідного коду логічно очікувати й на зменшення часу його написання, слід також очікувати й на зменшення кількості помилок, що також має позитивний вплив на ефективність розробки моделей процесів розповсюдження інфекційних захворювань.

Дано визначення метрики функціональності інструменту епідеміологічного моделювання, що дозволяє оцінити обсяг сценаріїв та функцій доступних користувачу.

Дано визначення метрики ефективності моделі, що дозволяє виконувати порівняння різних моделей та інструментів моделювання з точки зору їх обчислювальної ефективності. Ця метрика є інваріантною до кількості агентів, тривалості симуляції, її гранульованості та кількості одночасно залучених до обчислення моделі ядер центрального процесора.

Аналіз швидкодії результівних моделей, заданих мовою CTrace, ставив дві мети: обґрунтувати доцільність використання мови Python як проміжної мови

трансляції, а також порівняти їх швидкодію з наявними моделями. Показано, що використання мови Python з бібліотекою для JIT-компіляції Numba дозволяє отримати швидкодію моделей, порівняну з отриманою у разі використання мови Rust, ефективність якої порівняна з C/C++. При цьому, гнучкість мови Python дозволяє відносно просто вносити корективи у процес трансляції. Експерименти з використанням різних тестових середовищ показали, що існуючі епідеміологічні моделі не виграють у швидкодії у розроблених мовою CTrace еквівалентів. Так, у найгіршому випадку розроблений підхід програє найефективнішому на 15%, у найкращому – виграш у швидкості сягає 988%.

З результатів порівняння розробленого середовища CTraceEnv з аналогами за визначеними метриками ефективності та функціональності можна зробити висновок що обсяг функцій агентного епідеміологічного моделювання наданий CTraceEnv є значно більшим за аналоги й приблизно відповідає середовищу NetLogo, надаючи при цьому ефективність результівних моделей, що не програє аналогам.

ВИСНОВКИ

У дисертаційній роботі вирішена актуальна наукова проблема створення інструментального забезпечення для ефективного моделювання процесів розповсюдження інфекційних захворювань. Було отримано наступні наукові та практичні результати:

1. Опрацьована література за темою математичних, програмних та мовних засобів моделювання розповсюдження інфекційних захворювань з метою виокремлення основних дослідницьких напрямків та аналізу поточних рішень. Були окреслені основні підходи до епідеміологічного моделювання, а саме дослідження компартментних та агентних моделей, серед яких значна увага дослідницької спільноти спрямована на останні через їх гнучкість та точність. Окрім цього, були розглянути напрями розробки програмних засобів їх опису та аналітики. Проведений аналіз показав суттєвий недолік наявних інструментів моделювання, а саме відсутність засобів розробки агентних моделей, які б поєднували швидкість та простоту їх опису. Був зроблений висновок про доцільність розробки предметно-орієнтованої мови для опису агентних моделей.

2. Дано визначення формальної граматики мови опису агентних моделей розповсюдження інфекційних захворювань CTrace на основі розробленої загальної агентної епідеміологічної моделі;

3. Розроблений транслятор мови CTrace з Python або Rust як проміжної мови трансляції. Результатом роботи трансляції є бібліотека для мови Python, що надає програмний інтерфейс до необхідних методів роботи з моделлю. Були розроблені та експериментально перевірені оптимізації компілятора. Також, була досліджена ефективність використання мови Python як проміжної мови трансляції. Результати експериментів свідчать про те, що використання мови Python не призводить зо значного збільшення часу обрахунку моделі, зберігаючи при цьому можливість ефективного розширення транслятора;

4. Розроблене середовище розробки та аналізу агентних епідеміологічних

моделей з використанням мови CTrace – CTraceEnv. Розроблене середовище надає базовий функціонал необхідний для підтримки процесу опису моделей мовою CTrace, що включає підсвітку синтаксису вихідного коду, вбудований транслятор тощо. Серед функціоналу аналізу представлені елементи керування роботою моделі, значень її глобальних параметрів, інтерфейс перегляду динаміки розповсюдження досліджуваного інфекційного захворювання, а також функціонал експорту результатів для подальшого аналізу сторонніми спеціалізованими інструментами.

5. Виконана практична апробація розробленого інструментарію для ефективного моделювання розповсюдження інфекційних захворювань. У її рамках, було дано визначення агентної епідеміологічної моделі розповсюдження коронавірусу SARS-CoV-2 серед населення Польщі у період з початку вересня 2020 року до кінця листопада 2020 року, що була відкалібрована з використанням публічно доступних епідеміологічних даних. Результівна модель описує динаміку зміни кількості інфікованих людей з коефіцієнтом детермінації рівним 0.9319, що свідчить про здатність розробленого підходу описувати процеси розповсюдження інфекційних захворювань.

6. Виконаний порівняльний аналіз середовища CTraceEnv як підходу до розробки ефективних моделей розповсюдження інфекційних захворювань з наявними аналогами. Якісний аналіз показав, що обсяг сценаріїв, що можуть бути досліджені з використанням заданих мовою CTrace моделей є більшим за той, що надають інші інструменти моделювання, за вийнятком платформи NetLogo та мов програмування загального призначення. Аналіз ефективності роботи моделей заданих мовою CTrace показав, що час обчислення кроку їх симуляції не програє іншим наявним моделям, випереджаючи зокрема платформу NetLogo на три порядки. Так, у найгіршому випадку розроблений підхід програє найефективнішому на 15%, у найкращому – виграш у швидкості сягає 988%. Опосередкованою ознакою значної переваги розробленого інструментарію є зменшення

обсягу вихідного коду моделі у 14-34 рази у порівнянні з мовою Python та 16-45 разів з мовою Rust. Окрім того, що зі зменшенням обсягу вихідного коду логічно очікувати й на зменшення часу його написання, слід також очікувати й на зменшення кількості помилок, що також має позитивний вплив на ефективність розробки моделей процесів розповсюдження інфекційних захворювань.

СПИСОК ЛІТЕРАТУРИ

[1] Cyrus H Gordon. “The new Amarna tablets”. В: *Orientalia* 16.1 (1947), с. 1—21.

[2] James P Warbasse. “Anomalies and Curiosities of Medicine”. В: *Annals of Surgery* 26.1 (1897), с. 150.

[3] Suzanne Austin Alchon. *A pest in the land: new world epidemics in a global perspective*. UNM Press, 2003, с. 21.

[4] William Easterly. “Review of Walter Scheidel’s The Great Leveler: Violence and the History of Inequality from the Stone Age to the Twenty-First Century”. В: *Journal of Economic Literature* 57.4 (2019), с. 292—293.

[5] Joseph A Legan. “The medical response to the Black Death”. В: *The medical response to the Black Death “by Joseph A. Legan (jmu. edu)* (2015), с. 6.

[6] Coronaviridae Study Group of the International Committee on Taxonomy of Viruses та ін. “The species Severe acute respiratory syndrome-related coronavirus: classifying 2019-nCoV and naming it SARS-CoV-2”. В: *Nature microbiology* 5.4 (2020), с. 536. DOI: 10.1038/s41564-020-0695-z.

[7] Fabrizio Carinci. “Covid-19: preparedness, decentralisation, and the hunt for patient zero”. В: *Bmj* 368 (2020).

[8] Youngmee Jee. “WHO international health regulations emergency committee for the COVID-19 outbreak”. В: *Epidemiology and health* 42 (2020).

[9] Raj Dandekar та George Barbastathis. “Quantifying the effect of quarantine control in Covid-19 infectious spread using machine learning”. В: *medRxiv* (2020).

[10] Alan D Kaye та ін. “Economic impact of COVID-19 pandemic on healthcare facilities and systems: International perspectives”. В: *Best Practice & Research Clinical Anaesthesiology* 35.3 (2021), с. 293—306.

[11] Fatma Altuntas та Mehmet Sahin Gok. “The effect of COVID-19 pandemic on domestic tourism: A DEMATEL method analysis on quarantine decisions”. В: *International Journal of Hospitality Management* 92 (2021), с. 102719.

[12] Yunhe Wang та ін. “The impact of quarantine on mental health status among general population in China during the COVID-19 pandemic”. В: *Molecular psychiatry* (2021), с. 1—10.

[13] Megan L Ranney, Valerie Griffeth та Ashish K Jha. “Critical supply shortages - the need for ventilators and personal protective equipment during the Covid-19 pandemic”. В: *New England Journal of Medicine* 382.18 (2020), e41.

[14] Preeti Mehrotra, Preeti Malani та Prashant Yadav. “Personal protective equipment shortages during COVID-19—supply chain–related causes and mitigation strategies”. В: *JAMA Health Forum*. Т. 1. 5. American Medical Association. 2020.

[15] Gary Gereffi. “What does the COVID-19 pandemic teach us about global value chains? The case of medical supplies”. В: *Journal of International Business Policy* 3.3 (2020), с. 287—301.

[16] ВВ Сарнацький та ІВ Баклан. “ВПЛИВ СОЦІОДЕМОГРАФІЧНОЇ ГЕТЕРОГЕННОСТІ НА ОПТИМАЛЬНУ СТРАТЕГІЮ ВПРОВАДЖЕННЯ КАРАНТИННИХ ЗАХОДІВ”. В: *Вчені записки Таврійського національного університету. Серія Технічні науки* 32.4 (2021), с. 149—155.

[17] ВВ Сарнацький та ІВ Баклан. “ВПЛИВ СОЦІОДЕМОГРАФІЧНОЇ ГЕТЕРОГЕННОСТІ НА ОПТИМАЛЬНУ СТРАТЕГІЮ ВПРОВАДЖЕННЯ КАРАНТИННИХ ЗАХОДІВ”. В: (2021).

[18] William Ogilvy Kermack та Anderson G McKendrick. “A contribution to the mathematical theory of epidemics”. В: *Proceedings of the royal society*

of london. Series A, Containing papers of a mathematical and physical character 115.772 (1927), с. 700—721.

[19] Николай Сергеевич Бахвалов, Николай Петрович Жидков та Георгий Михайлович Кобельков. *Численные методы*. М. Лаборатория Базовых Знаний, 2001. ISBN: 5-93208-043-4.

[20] Konstantin S Sharov. “Creating and applying SIR modified compartmental model for calculation of COVID-19 lockdown efficiency”. В: *Chaos, Solitons & Fractals* 141 (2020), с. 110295.

[21] Shipeng Nie та Weide Li. “Using lattice SIS epidemiological model with clustered treatment to investigate epidemic control”. В: *Biosystems* 191 (2020), с. 104119.

[22] Alberto Ceria та ін. “Modeling airport congestion contagion by heterogeneous SIS epidemic spreading on airline networks”. В: *Plos one* 16.1 (2021), e0245043.

[23] Wahidah Sanusi та ін. “Analysis and Simulation of SIRS Model for Dengue Fever Transmission in South Sulawesi, Indonesia”. В: *Journal of Applied Mathematics* 2021 (2021).

[24] Jun Inamo. “How should we overcome the threat by the pandemic of 2019-nCoV? Epidemic simulation using the SIRS model”. В: 39 (2022).

[25] James D Cherry. “The history of pertussis (whooping cough); 1906–2015: facts, myths , and misconceptions”. В: *Current Epidemiology Reports* 2.2 (2015), с. 120—130.

[26] Kimia Ameri та Kathryn D Cooper. “A Network-Based compartmental model for the spread of whooping cough in Nebraska”. В: *AMIA Summits on Translational Science Proceedings* 2019 (2019), с. 388.

[27] Hee-Young Shin. “A multi-stage SEIR (D) model of the COVID-19 epidemic in Korea”. B: *Annals of Medicine* 53.1 (2021), с. 1160—1170.

[28] Suwardi Annas та ін. “Stability analysis and numerical simulation of SEIR model for pandemic COVID-19 spread in Indonesia”. B: *Chaos, Solitons & Fractals* 139 (2020), с. 110072.

[29] Zubair Ahmad та ін. “A report on COVID-19 epidemic in Pakistan using SEIR fractional model”. B: *Scientific Reports* 10.1 (2020), с. 1—14.

[30] Wenning Li та ін. “An evaluation of COVID-19 transmission control in Wenzhou using a modified SEIR model”. B: *Epidemiology & Infection* 149 (2021).

[31] Michael W Levin, Mingfeng Shang та Raphael Stern. “Effects of short-term travel on COVID-19 spread: A novel SEIR model and case study in Minnesota”. B: *Plos one* 16.1 (2021), e0245919.

[32] Eva K Lee, Yifan Liu та Ferdinand H Pietz. “A compartmental model for Zika virus with dynamic human and vector populations”. B: *AMIA Annual Symposium Proceedings*. T. 2016. American Medical Informatics Association. 2016, с. 743.

[33] Anthony S Fauci та David M Morens. “Zika virus in the Americas—yet another arbovirus threat”. B: *New England journal of medicine* 374.7 (2016), с. 601—604.

[34] Kent A Sepkowitz. “AIDS—the first 20 years”. B: *New England Journal of Medicine* 344.23 (2001), с. 1764—1772.

[35] Z Lu та ін. “A mathematical model for HIV prevention and control among men who have sex with men in China”. B: *Epidemiology & Infection* 148 (2020).

[36] Ernesto P Esteban та Lusmeralis Almodovar-Abreu. “Assessing the impact of vaccination in a COVID-19 compartmental model”. В: *Informatics in Medicine Unlocked* 27 (2021), с. 100795.

[37] Özden O Dalgıç та ін. “Deriving effective vaccine allocation strategies for pandemic influenza: Comparison of an agent-based simulation and a compartmental model”. В: *PloS one* 12.2 (2017), e0172261.

[38] Samuel Mwalili та ін. “SEIR model for COVID-19 dynamics incorporating the environment and social distancing”. В: *BMC Research Notes* 13.1 (2020), с. 1—5.

[39] George Macdonald. “The analysis of equilibrium in malaria”. В: *Tropical Disease Bulletin* 49.9 (1952), с. 813—829.

[40] Dirk Eisinger та Hans-Hermann Thulke. “Spatial pattern formation facilitates eradication of infectious diseases”. В: *The Journal of applied ecology* 45.2 (2008), с. 415. DOI: 10.1111/j.1365-2664.2007.01439.x.

[41] Steven F Railsback та Volker Grimm. *Agent-based and individual-based modeling: a practical introduction*. Princeton university press, 2019.

[42] M Elizabeth Halloran та ін. “Containing bioterrorist smallpox”. В: *Science* 298.5597 (2002), с. 1428—1432.

[43] Megan Murray. “Determinants of cluster distribution in the molecular epidemiology of tuberculosis”. В: *Proceedings of the National Academy of Sciences* 99.3 (2002), с. 1538—1543.

[44] Neil M Ferguson та ін. “Strategies for containing an emerging influenza pandemic in Southeast Asia”. В: *Nature* 437.7056 (2005), с. 209—214.

[45] Philip Cooley та ін. “Protecting health care workers: a pandemic simulation based on Allegheny County”. В: *Influenza and other respiratory viruses* 4.2 (2010), с. 61—72.

[46] Wayne TA Enanoria та ін. “The effect of contact investigations and public health interventions in the control and prevention of measles transmission: A simulation study”. В: *PloS one* 11.12 (2016), e0167160.

[47] Md Zahangir Alam та ін. “A spatial agent-based model of Anopheles vagus for malaria epidemiology: examining the impact of vector control interventions”. В: *Malaria journal* 16.1 (2017), с. 1—20.

[48] Mariusz Maziarz та Martin Zach. “Agent-based modelling for SARS-CoV-2 epidemic prediction and intervention assessment: A methodological appraisal”. В: *Journal of Evaluation in Clinical Practice* 26.5 (2020), с. 1352—1360.

[49] Erik Cuevas. “An agent-based model to evaluate the COVID-19 transmission risks in facilities”. В: *Computers in biology and medicine* 121 (2020), с. 103827.

[50] Jonatan Gomez та ін. “INFEKTA—An agent-based model for transmission of infectious diseases: The COVID-19 case in Bogotá, Colombia”. В: *PloS one* 16.2 (2021), e0245787.

[51] KK Avilov та O Yu Solovey. “Agent-Based Models: Approaches and Applicability to Epidemiology”. В: *Matematicheskaya Biologiya i Bioinformatika* 7 (2 2012), с. 425—443.

[52] Nicholson Collier та North Michael. “Parallel Agent-Based Simulation with Repast for High Performance Computing”. В: *SIMULATION* 89 (10 2013).

[53] Pietro Terna та ін. “Simulation tools for social scientists: Building agent based models with swarm”. В: *Journal of artificial societies and social simulation* 1.2 (1998), с. 1—12.

[54] Ferdinando Chiacchio та ін. “Agent-based modeling of the immune system: NetLogo, a promising framework”. В: *BioMed research international* 2014 (2014).

[55] DANIEL D KITAY. “The Oracle DBMS Architecture: A Technical Introduction”. В: *Technical Support* (1997).

[56] Stefano Merler та Marco Ajelli. “The role of population heterogeneity and human mobility in the spread of pandemic influenza”. В: *Proceedings of the Royal Society B: Biological Sciences* 277.1681 (2010), с. 557—565.

[57] Eurostat. European Commission. URL: https://ec.europa.eu/eurostat (дата звернення: 15.09.2022).

[58] Amir Mokhtari та Jane M Van Doren. “An agent-based model for pathogen persistence and cross-contamination dynamics in a food facility”. В: *Risk Analysis* 39.5 (2019), с. 992—1021.

[59] World Health Organization та ін. *Water, sanitation, hygiene, and waste management for the COVID-19 virus: interim guidance, 23 April 2020*. Тех. звіт. World Health Organization, 2020.

[60] *Information technology — Vocabulary*. Standard. International Organization for Standardization, 2015.

[61] Алексей Всеволодович Гладкий. *Формальные грамматики и языки*. Рипол Классик, 1973.

[62] William F Schmitt. “The Univac Short Code”. В: *Annals of the History of Computing* 10.1 (1988).

[63] Eurostat. European Commission. URL: https://ec.europa.eu/eurostat (дата звернення: 03.09.2022).

[64] D Pigott. *Online Historical Encyclopaedia of Programming Languages*. 2020. URL: https://hopl.info. (дата звернення: 05.09.2022).

[65] Евгений Всеволодович Роганов. *Основы информатики и программирования*. МГИУ, 2002, с. 315. ISBN: 5-276-00187-1.

[66] Ricardo Marroquim та André Maximo. “Introduction to GPU Programming with GLSL”. В: *2009 Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing*. IEEE. 2009, с. 3—16.

[67] Tim Berglund та Matthew McCullough. *Building and testing with Gradle*. ”O’Reilly Media, Inc.”, 2011.

[68] Mai Anh BUI T та ін. “The Kendrick modelling platform: language abstractions and tools for epidemiology”. В: *BMC bioinformatics* 20.1 (2019), с. 1—13.

[69] Wolfram Research Inc. *Mathematica, Version 13.0.0*. URL: https://www.wolfram.com/mathematica. (дата звернення: 05.09.2022).

[70] William Schelter. *Maxima, Version 5.45.1*. URL: https://maxima.sourceforge.io/. (дата звернення: 05.09.2022).

[71] MATLAB. *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[72] Modelica Association. *Modelica, Version 3.3*. URL: https://www.modelica.org/. (дата звернення: 05.09.2022).

[73] Scilab Enterprise. *Scilab, Version 6.1.0*. URL: https://www.scilab.org/. (дата звернення: 05.09.2022).

[74] Dennis L Chao та ін. “FluTE, a publicly available stochastic influenza epidemic simulation model”. В: *PLoS computational biology* 6.1 (2010), e1000656.

[75] Wouter Van den Broeck та ін. “The GLEaMviz computational tool, a publicly available software to explore realistic epidemic spreading scenarios at the global scale”. В: *BMC infectious diseases* 11.1 (2011), с. 1—14.

[76] Alexander Falenski та ін. “A generic open-source software framework supporting scenario simulations in bioterrorist crises”. В: *Biosecurity and bioterrorism: biodefense strategy, practice, and science* 11.S1 (2013), S134—S145.

[77] John J Grefenstette та ін. “FRED (A Framework for Reconstructing Epidemic Dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations”. В: *BMC public health* 13.1 (2013), с. 1—14.

[78] Dinesh P Mehta та Sartaj Sahni. “Handbook of data structures and applications”. В: Chapman та Hall/CRC, 2004. Гл. 9: Hash Tables, с. 163. DOI: https://doi.org/10.1201/9781420035179.

[79] Dinesh P Mehta та Sartaj Sahni. “Handbook of data structures and applications”. В: Chapman та Hall/CRC, 2004. Гл. 2.3: Linked Lists, с. 47. DOI: https://doi.org/10.1201/9781420035179.

[80] Tom Van Dijk. “Analysing and improving hash table performance”. В: *10th Twente Student Conference on IT. University of Twente, Faculty of Electrical Engineering and Computer Science*. Citeseer. 2009.

[81] Noam Chomsky. “Three models for the description of language”. В: *IRE Transactions on information theory* 2.3 (1956), с. 113—124. DOI: 10.1109/TIT.1956.1056813.

[82] Backus JW. “c The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference”. В: *Proceedings of the International Conference of Information Processing UNESCO Paris June*. 1959.

[83] *Information technology — Syntactic metalanguage — Extended BNF*. Standard. International Organization for Standardization, 1996.

[84] Warren Gilchrist. *Statistical modelling with quantile functions*. Chapman та Hall/CRC, 2000.

[85] *Jupyter*. URL: https://jupyter.org/. (дата звернення: 05.09.2022).

[86] Jiawei Wang, Li Li та Andreas Zeller. “Better code, better sharing: on the need of analyzing jupyter notebooks”. В: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*. 2020, с. 53—56.

[87] Siu Kwan Lam, Antoine Pitrou та Stanley Seibert. “Numba: A llvm-based python jit compiler”. В: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, с. 1—6.

[88] David Beazley. *Python Lex-Yacc*. 2001. URL: https://www.dabeaz.com/ply/. (дата звернення: 05.09.2022).

[89] Agner Fog та ін. “Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs”. В: *Copenhagen University College of Engineering* 93 (2011).

[90] Iyad Zayour та Hassan Hajjdiab. “How much integrated development environments (ides) improve productivity?” В: *J. Softw.* 8.10 (2013), с. 2425—2431.

[91] *PyQtGraph - Scientific Graphics and GUI Library for Python*. URL: https://www.pyqtgraph.org/. (дата звернення: 05.09.2022).

[92] *Riverbank Computing | Introduction*. URL: https://riverbankcomputing.com/software/pyqt/intro. (дата звернення: 05.09.2022).

[93] Volodymyr Mnih та ін. “Human-level control through deep reinforcement learning”. В: *nature* 518.7540 (2015), с. 529—533. DOI: 10 . 1038 / nature14236.

[94] Richard S Sutton та Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[95] Richard S Sutton та Andrew G Barto. “Reinforcement learning: An introduction”. В: MIT press, 2018, с. 47.

[96] Xavier Glorot, Antoine Bordes та Yoshua Bengio. “Deep sparse rectifier neural networks”. В: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop та Conference Proceedings. 2011, с. 315—323.

[97] Diederik P. Kingma та Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[98] *Worldometer*. URL: https : / / www . worldometers . info/. (дата звернення: 05.09.2022).

[99] Shuhei Watanabe. “Tree-structured Parzen estimator: Understanding its algorithm components and their roles for better empirical performance”. В: *arXiv preprint arXiv:2304.11127* (2023). DOI: 10.48550/arXiv.2304.11127.

[100] Miquel S. Porta. В: *A dictionary of epidemiology*. Oxford University Press, 2014, с. 289. ISBN: 9780199390069. DOI: 10.1093/acref/9780199976720.001.0001.

[101] Ben Lopman та ін. “Host, weather and virological factors drive norovirus epidemiology: time-series analysis of laboratory surveillance data in England and Wales”. В: *PloS one* 4.8 (2009). DOI: 10.1371/journal.pone.0006671.

[102] Giuseppe Giacopelli та ін. “A Full-Scale Agent-Based Model to Hypothetically Explore the Impact of Lockdown, Social Distancing, and Vaccination During the COVID-19 Pandemic in Lombardy, Italy: Model Development”. В: *Jmirx med* 2.3 (2021), e24630.

[103] Cristina Montañola-Sales та ін. “Modeling tuberculosis in Barcelona. A solution to speed-up agent-based simulations”. В: *2015 Winter Simulation Conference (WSC)*. IEEE. 2015, с. 1295—1306.

ДОДАТОК А. Опис тестової моделі мовою CTrace та проміжними мовами у процесі трансляції

Мова CTrace

```
gompertz(c, loc, scale) {
    icdf(p) = ln(1-ln(1-p)/c)*scale+loc;
}
gaussian(x, mu, sigma) =
    exp(-1 * ((x - mu) / sigma) ^ 2);
U(low, high) {
    icdf(p) = low + p * (high - low);
}

P_has_work(age) {
    icdf(p) = p < gaussian(
        age,
        41.56060857,
        21.78168914
    );
}

D_work_size {
    P(5, 10) = 0.46;
    P(10, 25) = 0.32;
    P(25, 50) = 0.125;
    P(50, 100) = 0.05;
    P(100, 200) = 0.02;
    P(200, 500) = 0.015;
    P(500, 1000) = 0.01;
}

D_closed_init {
    P(0) = 1;
    P(1) = 0;
}
D_house_size {
    P(2 ) = 0.571033;
    P(3 ) = 0.222209;
    P(4 ) = 0.156175;
```

```
36      P(5 ) = 0.039507;
37      P(6 ) = 0.008248;
38      P(7 ) = 0.001891;
39      P(8 ) = 0.000567;
40      P(9 ) = 0.000209;
41      P(10) = 0.000085;
42  }
43  $compartment
44  Work[10000] {
45      size ~ D_work_size;
46      closed ~ D_closed_init;
47  }
48  P_workplace(Work) {
49      prob(w) = size;
50  }
51  $compartment
52  Household[100000] {
53      size ~ D_house_size;
54      closed ~ D_closed_init;
55  }
56  P_household(Household) {
57      prob(h) = size;
58  }
59
60  $agent
61  Person[1000000] {
62      age ~ gompertz(
63          0.20510679560362763,
64          -1.82765672236963,
65          29.342139434877467
66      );
67      work ~ P_workplace;
68      has_work ~ P_has_work(age);
69      house ~ P_household;
70
71      schedule(h) = work.closed * house + (1 - work.closed) * (work * (h > 7) +
            house * (h <= 7));
72  }
73  A = 1;
74  Flu {
```

```
    S, E, I, R;

    P(S -> E | E) = 0.0002;
    P(S -> E | I) = 0.0005;

    P(S -> E) = 0.01;
    P(E -> I, 14) = 0.9;
    P(I -> R, 14) = 1;
}
```

Мова Python

```
import numpy as np
from numba import njit
from random import uniform
from base64 import b64decode
import numpy as functions


M=np.zeros(shape=(7060192,), dtype=np.float64)

@njit
def log(x): return np.log(x)
@njit
def log2(x): return np.log2(x)
@njit
def exp(x): return np.exp(x)
@njit
def sin(x): return np.sin(x)

@njit
def bisect(arr, left, right, x):
    if x < arr[left]: return left
    while left <= right:
        mid = (left+right) >> 1
        if arr[mid] > x: right = mid-1
        elif arr[mid] < x: left = mid+1
        else: return mid
    return left

```

```
29
30 @njit
31 def _gompertz_icdf(M):
32     M[8]=1.0-M[4]
33     M[9]=log(M[8],)
34     M[10]=M[9]/M[5]
35     M[11]=1.0-M[10]
36     M[12]=log(M[11],)
37     M[13]=M[12]*M[7]
38     M[14]=M[13]+M[6]
39
40 @njit
41 def gompertz_icdf(M,p,c,loc,scale):
42     M[4]=p
43     M[5]=c
44     M[6]=loc
45     M[7]=scale
46     _gompertz_icdf(M)
47     return M[14]
48
49
50 @njit
51 def _gompertz_sample(M):
52     M[18]=uniform(0.0,1.0)
53     M[4]=M[18]
54     M[5]=M[15]
55     M[6]=M[16]
56     M[7]=M[17]
57     _gompertz_icdf(M)
58     M[19]=M[14]
59
60 @njit
61 def gompertz_sample(M,c,loc,scale):
62     M[15]=c
63     M[16]=loc
64     M[17]=scale
65     _gompertz_sample(M)
66     return M[19]
67
68 @njit
```

```
69  def gompertz_sample_multi(M, start_addr, n_elements):
70      for i in range(n_elements):
71          M[start_addr + i] = gompertz_sample(M)
72
73  @njit
74  def _gaussian(M):
75      M[23]=M[20]-M[21]
76      M[24]=M[23]/M[22]
77      M[25]=M[24]**2.0
78      M[26]=-1.0*M[25]
79      M[27]=exp(M[26],)
80
81  @njit
82  def gaussian(M,x,mu,sigma):
83      M[20]=x
84      M[21]=mu
85      M[22]=sigma
86      _gaussian(M)
87      return M[27]
88
89
90  @njit
91  def _U_icdf(M):
92      M[33]=M[32]-M[31]
93      M[34]=M[30]*M[33]
94      M[35]=M[31]+M[34]
95
96  @njit
97  def U_icdf(M,p,low,high):
98      M[30]=p
99      M[31]=low
100     M[32]=high
101     _U_icdf(M)
102     return M[35]
103
104
105 @njit
106 def _U_sample(M):
107     M[38]=uniform(0.0,1.0)
108     M[30]=M[38]
```

```
109     M[31]=M[36]
110     M[32]=M[37]
111     _U_icdf(M)
112     M[39]=M[35]
113 
114 @njit
115 def U_sample(M,low,high):
116     M[36]=low
117     M[37]=high
118     _U_sample(M)
119     return M[39]
120 
121 @njit
122 def U_sample_multi(M, start_addr, n_elements):
123     for i in range(n_elements):
124         M[start_addr + i] = U_sample(M)
125 
126 @njit
127 def _P_has_work_icdf(M):
128     M[20]=M[42]
129     M[21]=41.56060857
130     M[22]=21.78168914
131     _gaussian(M)
132     M[43]=M[27]
133     M[44]=M[41]<M[43]
134 
135 @njit
136 def P_has_work_icdf(M,p,age):
137     M[41]=p
138     M[42]=age
139     _P_has_work_icdf(M)
140     return M[44]
141 
142 
143 @njit
144 def _P_has_work_sample(M):
145     M[46]=uniform(0.0,1.0)
146     M[41]=M[46]
147     M[42]=M[45]
148     _P_has_work_icdf(M)
```

```
149     M[47]=M[44]
150 
151 @njit
152 def P_has_work_sample(M, age):
153     M[45]=age
154     _P_has_work_sample(M)
155     return M[47]
156 
157 @njit
158 def P_has_work_sample_multi(M, start_addr, n_elements):
159     for i in range(n_elements):
160         M[start_addr + i] = P_has_work_sample(M)
161 @njit
162 def D_work_size_sample(M,):
163     _D_work_size_sample(M)
164     return M[48]
165 
166 
167 @njit
168 def _D_work_size_sample(M):
169     M[56]=bisect(M,49,55,uniform(0.0, 1.0))
170     M[57]=M[56]-49.0
171     ch=M[57]
172     if ch == 0.0:
173         M[48]=uniform(5.0,10.0)
174     elif ch == 1.0:
175         M[48]=uniform(10.0,25.0)
176     elif ch == 2.0:
177         M[48]=uniform(25.0,50.0)
178     elif ch == 3.0:
179         M[48]=uniform(50.0,100.0)
180     elif ch == 4.0:
181         M[48]=uniform(100.0,200.0)
182     elif ch == 5.0:
183         M[48]=uniform(200.0,500.0)
184     else:
185         M[48]=uniform(500.0,1000.0)
186 
187 @njit
188 def D_work_size_sample_multi(M, start_addr, n_elements):
```

```
    for i in range(n_elements):
        M[start_addr + i] = D_work_size_sample(M)
@njit
def D_closed_init_sample(M,):
    _D_closed_init_sample(M)
    return M[58]


@njit
def _D_closed_init_sample(M):
    M[61]=bisect(M,59,60,uniform(0.0, 1.0))
    M[62]=M[61]-59.0
    ch=M[62]
    M[58]=ch+0.0

@njit
def D_closed_init_sample_multi(M, start_addr, n_elements):
    for i in range(n_elements):
        M[start_addr + i] = D_closed_init_sample(M)
@njit
def D_house_size_sample(M,):
    _D_house_size_sample(M)
    return M[63]


@njit
def _D_house_size_sample(M):
    M[73]=bisect(M,64,72,uniform(0.0, 1.0))
    M[74]=M[73]-64.0
    ch=M[74]
    M[63]=ch+2.0

@njit
def D_house_size_sample_multi(M, start_addr, n_elements):
    for i in range(n_elements):
        M[start_addr + i] = D_house_size_sample(M)

@njit
def _Work_step(M):
    pass
```

```
229
230 @njit
231 def Work_size(M):
232     return M[76:30076]
233 @njit
234 def Work_closed(M):
235     return M[30078:60078]
236
237 @njit
238 def _P_workplace_prob(M):
239     pass
240
241 @njit
242 def P_workplace_prob(M, size, closed):
243     M[60080]=size
244     M[60081]=closed
245     _P_workplace_prob(M)
246     return M[60080]
247
248
249 @njit
250 def _P_workplace_sample(M):
251     probs = np.zeros(shape=(30000,))
252     for M[60082] in range(30000):
253         M[60080]=M[int(76.0 + M[60082])]
254         M[60081]=M[int(30078.0 + M[60082])]
255         _P_workplace_prob(M)
256         M[60085]=M[60080]
257         probs[int(M[60082])]=M[60085]
258     pass
259     probs = probs.cumsum()
260     probs = probs / probs[-1]
261     M[60086]=uniform(0.0,1.0)
262     M[60087]=bisect(probs,0,30000,M[60086])
263     M[60088]=M[60087]+0.0
264
265 @njit
266 def P_workplace_sample(M,):
267     _P_workplace_sample(M)
268     return M[60088]
```

```
269
270 @njit
271 def P_workplace_sample_multi(M, start_addr ,n_samples):
272     probs = np.zeros(shape=(30000,))
273     for M[60082] in range(30000):
274         M[60080]=M[int(76.0 + M[60082])]
275         M[60081]=M[int(30078.0 + M[60082])]
276         _P_workplace_prob(M)
277         M[60085]=M[60080]
278         probs[int(M[60082])]=M[60085]
279     pass
280     probs = probs.cumsum()
281     probs = probs / probs[-1]
282     for i in range(n_samples):
283         M[int(start_addr + i)]=0.0 + bisect(probs, 0, 30000, uniform(0.0, 1.0))
284     pass
285     return 0.0
286
287
288 @njit
289 def _Household_step(M):
290     pass
291
292 @njit
293 def Household_size(M):
294     return M[60090:560090]
295 @njit
296 def Household_closed(M):
297     return M[560092:1060092]
298
299 @njit
300 def _P_household_prob(M):
301     pass
302
303 @njit
304 def P_household_prob(M, size ,closed):
305     M[1060094]=size
306     M[1060095]=closed
307     _P_household_prob(M)
308     return M[1060094]
```

```
309
310
311 @njit
312 def _P_household_sample(M):
313     probs = np.zeros(shape=(500000,))
314     for M[1060096] in range(500000):
315         M[1060094]=M[int(60090.0 + M[1060096])]
316         M[1060095]=M[int(560092.0 + M[1060096])]
317         _P_household_prob(M)
318         M[1060099]=M[1060094]
319         probs[int(M[1060096])]=M[1060099]
320     pass
321     probs = probs.cumsum()
322     probs = probs / probs[-1]
323     M[1060100]=uniform(0.0,1.0)
324     M[1060101]=bisect(probs,0,500000,M[1060100])
325     M[1060102]=M[1060101]+30000.0
326
327 @njit
328 def P_household_sample(M,):
329     _P_household_sample(M)
330     return M[1060102]
331
332 @njit
333 def P_household_sample_multi(M, start_addr, n_samples):
334     probs = np.zeros(shape=(500000,))
335     for M[1060096] in range(500000):
336         M[1060094]=M[int(60090.0 + M[1060096])]
337         M[1060095]=M[int(560092.0 + M[1060096])]
338         _P_household_prob(M)
339         M[1060099]=M[1060094]
340         probs[int(M[1060096])]=M[1060099]
341     pass
342     probs = probs.cumsum()
343     probs = probs / probs[-1]
344     for i in range(n_samples):
345         M[int(start_addr + i)]=30000.0 + bisect(probs, 0, 500000, uniform(0.0,
                1.0))
346     pass
347     return 0.0
```

```
348
349
350 @njit
351 def _Person_step(M):
352     pass
353
354 @njit
355 def Person_age(M):
356     return M[1060104:2060104]
357 @njit
358 def Person_work(M):
359     return M[2060106:3060106]
360 @njit
361 def Person_has_work(M):
362     return M[3060108:4060108]
363 @njit
364 def Person_house(M):
365     return M[4060110:5060110]
366 @njit
367 def Person_inf_state(M):
368     return M[5060126:6060126]
369 @njit
370 def Person_inf_days(M):
371     return M[6060126:7060126]
372
373 @njit
374 def _Person_schedule(M):
375     M[5060116]=M[int(30078.0 + M[int(2060106.0 + M[5060114])])]*M[int(4060110.0
        + M[5060114])]
376     M[5060118]=1.0-M[int(30078.0 + M[int(2060106.0 + M[5060114])])]
377     M[5060119]=M[5060113]>7.0
378     M[5060120]=M[int(2060106.0 + M[5060114])]*M[5060119]
379     M[5060121]=M[5060113]<=7.0
380     M[5060122]=M[int(4060110.0 + M[5060114])]*M[5060121]
381     M[5060123]=M[5060120]+M[5060122]
382     M[5060124]=M[5060118]*M[5060123]
383     M[5060125]=M[5060116]+M[5060124]
384
385 @njit
386 def Person_schedule(M,h,i):
```

```
387     M[5060113]=h
388     M[5060114]=i
389     _Person_schedule(M)
390     return M[5060125]
391
392 @njit
393 def Flu_state_prob_sample(M,):
394     _Flu_state_prob_sample(M)
395     return M[7060127]
396
397
398 @njit
399 def _Flu_state_prob_sample(M):
400     M[7060132]=bisect(M,7060128,7060131,uniform(0.0, 1.0))
401     M[7060133]=M[7060132]-7060128.0
402     ch=M[7060133]
403     M[7060127]=ch+0
404
405 @njit
406 def Flu_state_prob_sample_multi(M, start_addr, n_elements):
407     for i in range(n_elements):
408         M[start_addr + i] = Flu_state_prob_sample(M)
409 @njit
410 def _Flu_progress_prob(M,out):
411     s = M[7060139]
412     d = M[7060140]
413     out[:] = 0
414     if s == 0:
415         if True:
416             out[1] = -0.014499569695115089
417             return
418     elif s == 1:
419         if d == 14.0:
420             out[2] = -3.3219280948873626
421             return
422     elif s == 2:
423         if d == 14.0:
424             out[3] = -39.86313713864835
425             return
426 @njit
```

```
427 def Flu_progress_prob(M, inf_state, inf_days, out):
428     M[7060139]=inf_state
429     M[7060140]=inf_days
430     _Flu_progress_prob(M, out)
431 @njit
432 def _Flu_spread_prob(M, out):
433     sd = M[int(5060126.0 + M[7060154])]
434     if sd == 1:
435         out[0,1] += -0.0002885678659263426
436     elif sd == 2:
437         out[0,1] += -0.0007215279174593579
438 @njit
439 def Flu_spread_prob(M, agent_r, agent_d, comp_i, out):
440     M[7060153]=agent_r
441     M[7060154]=agent_d
442     M[7060155]=comp_i
443     _Flu_spread_prob(M, out)
444 @njit
445 def _Flu_spread_prob_inc(M, out):
446     sr = M[int(5060126.0 + M[7060153])]
447     sd = M[int(5060126.0 + M[7060154])]
448     if sd == 1:
449         if sr == 0:
450             out[1] += -0.0002885678659263426
451         elif sr == 1:
452             pass
453         elif sr == 2:
454             pass
455         elif sr == 3:
456             pass
457     elif sd == 2:
458         if sr == 0:
459             out[1] += -0.0007215279174593579
460         elif sr == 1:
461             pass
462         elif sr == 2:
463             pass
464         elif sr == 3:
465             pass
466 @njit
```

```
467 def Flu_spread_prob_inc(M, agent_r, agent_d, comp_i, out):
468     M[7060153]=agent_r
469     M[7060154]=agent_d
470     M[7060155]=comp_i
471     _Flu_spread_prob_inc(M, out)
472 @njit
473 def Flu_spread_0_0_branch_sample(M,):
474     _Flu_spread_0_0_branch_sample(M)
475     return M[7060170]
476 
477 
478 @njit
479 def _Flu_spread_0_0_branch_sample(M):
480     M[7060175]=bisect(M,7060171,7060174,uniform(0.0, 1.0))
481     M[7060176]=M[7060175]-7060171.0
482     ch=M[7060176]
483     M[7060170]=ch+0
484 
485 @njit
486 def Flu_spread_0_0_branch_sample_multi(M, start_addr, n_elements):
487     for i in range(n_elements):
488         M[start_addr + i] = Flu_spread_0_0_branch_sample(M)
489 @njit
490 def Flu_spread_0_1_branch_sample(M,):
491     _Flu_spread_0_1_branch_sample(M)
492     return M[7060177]
493 
494 
495 @njit
496 def _Flu_spread_0_1_branch_sample(M):
497     M[7060182]=bisect(M,7060178,7060181,uniform(0.0, 1.0))
498     M[7060183]=M[7060182]-7060178.0
499     ch=M[7060183]
500     M[7060177]=ch+0
501 
502 @njit
503 def Flu_spread_0_1_branch_sample_multi(M, start_addr, n_elements):
504     for i in range(n_elements):
505         M[start_addr + i] = Flu_spread_0_1_branch_sample(M)
506 @njit
```

```
507 def Flu_is_susceptible(M,r):
508     M[7060153]=r
509     return M[int(M[int(5060126.0 + M[7060153])] + 7060184.0)]
510 
511 @njit
512 def Flu_is_infectious(M,d):
513     M[7060154]=d
514     return M[int(M[int(5060126.0 + M[7060154])] + 7060188.0)]
515 
516 probs = np.zeros((530000, 24, 4, 4), dtype=np.float64)
517 probs_altered = np.zeros((530000,), dtype=np.bool8)
518 @njit
519 def infection_spread(M, probs, probs_altered):
520     pr = np.empty((4, 4))
521     pp = np.empty(4)
522     for p in range(1000000):
523         if bool(Flu_is_infectious(M, p)):
524             old_c = -1
525             for h in range(24):
526                 c = int(Person_schedule(M, h, p))
527                 Flu_spread_prob(M, 0, p, c, probs[c, h])
528                 probs_altered[c] = True
529     old_prob = np.empty(4)
530     for p in range(1000000):
531         Flu_progress_prob(M, M[5060126 + p], M[6060126 + p], pp)
532         sd = int(M[5060126 + p])
533         if bool(Flu_is_susceptible(M, p)):
534             for i in range(4):
535                 old_prob[i] = 0
536             old_c = -1
537             for h in range(24):
538                 c = int(Person_schedule(M, h, p))
539                 if not probs_altered[c]: continue
540                 if c != old_c:
541                     for i in range(4): old_prob[i] = probs[c, h, sd, i]
542                     old_c = c
543                 pp += old_prob
544         sm = 0
545         for s in range(4):
546             if s == sd: continue
```

```
                pp[s] = 1 - 2 ** pp[s]
                sm += pp[s]
            pp[sd] = 1 - sm
            for i in range(1, 4): pp[i] += pp[i - 1]
            new_s = bisect(pp, 0, 3, uniform(0, 1))
            if new_s != sd:
                M[5060126 + p], M[6060126 + p] = new_s, 1
            else:
                M[6060126 + p] += 1
        for i in range(530000):
            if probs_altered[i]:
                probs_altered[i] = False
                probs[i] = 0

def step(M,):
    infection_spread(M,probs,probs_altered);
    _Work_step(M)
    _Household_step(M)
    _Person_step(M)
    M[0]=M[0] + 1.0
    return 0.0

@njit
def init(M,):
    for i in range(7060192):
        M[i] = 0
    M[0]=0
    M[49]=0.46
    M[50]=0.78
    M[51]=0.905
    M[52]=0.9550000000000001
    M[53]=0.9750000000000001
    M[54]=0.9900000000000001
    M[55]=1.0
    M[59]=1.0
    M[60]=1.0
    M[64]=0.5710764018065374
    M[65]=0.7933022909741141
    M[66]=0.9494891611762494
    M[67]=0.9889991639364591
```

```
    M[68]=0.9972477908321032
    M[69]=0.9991389345590265
    M[70]=0.9997059776543017
    M[71]=0.999914993539509
    M[72]=1.0
    for M[75] in range(30000):
        _D_work_size_sample(M)
        M[30076]=M[48]
        M[int(76.0 + M[75])]=M[30076]
        _D_closed_init_sample(M)
        M[60078]=M[58]
        M[int(30078.0 + M[75])]=M[60078]
        pass
    for M[60089] in range(500000):
        _D_house_size_sample(M)
        M[560090]=M[63]
        M[int(60090.0 + M[60089])]=M[560090]
        _D_closed_init_sample(M)
        M[1060092]=M[58]
        M[int(560092.0 + M[60089])]=M[1060092]
        pass
    P_workplace_sample_multi(M, 2060106, 1000000)
    P_household_sample_multi(M, 4060110, 1000000)
    for M[1060103] in range(1000000):
        M[15]=0.20510679560362763
        M[16]=-1.82765672236963
        M[17]=29.342139434877467
        _gompertz_sample(M)
        M[2060104]=M[19]
        M[int(1060104.0 + M[1060103])]=M[2060104]
        M[45]=M[int(1060104.0 + M[1060103])]
        _P_has_work_sample(M)
        M[4060108]=M[47]
        M[int(3060108.0 + M[1060103])]=M[4060108]
        pass
    M[7060126]=1.0
    M[7060128]=1.0
    M[7060129]=1.0
    M[7060130]=1.0
    M[7060131]=1.0
```

```
    M[7060171]=0.9998
    M[7060172]=1.0
    M[7060173]=1.0
    M[7060174]=1.0
    M[7060178]=0.9995
    M[7060179]=1.0
    M[7060180]=1.0
    M[7060181]=1.0
    M[7060184]=1
    M[7060185]=0
    M[7060186]=0
    M[7060187]=0
    M[7060188]=0
    M[7060189]=1
    M[7060190]=1
    M[7060191]=0
     Flu_state_prob_sample_multi(M, 5060126, 1000000)
     return 0.0

@njit
def set_A(M, val): M[7060126] = val
@njit
def get_A(M): return M[7060126]
PARAMS = ['A']
STATES = ['S','E','I','R']
```

Мова Rust

```
static mut M: Vec<f64> = Vec::<f64>::new();
static mut sc: Vec<[f64; 24]> = Vec::<[f64; 24]>::new();


static mut ch: f64 = 0.0;

fn exp(x: f64) -> f64 { x.exp() }
fn log(x: f64) -> f64 { x.ln() }

fn uniform(a: f64, b: f64) -> f64 {
    (b - a) * rand::random::<f64>() + a
}
```

```
13
14 fn bisect(arr: &[f64], left: usize, right: usize, x: f64) -> usize {
15     let mut left = left;
16     let mut right = right;
17     let mut mid = 0;
18     if x < arr[left] { return left };
19     while left <= right {
20         mid = (left + right) >> 1;
21         if arr[mid] > x { right = mid - 1}
22         else if arr[mid] < x { left = mid + 1}
23         else { return mid };
24     }
25     left
26 }
27
28 unsafe fn _gompertz_icdf() {
29     *M.get_unchecked_mut(8)=(1.0 - *M.get_unchecked(4) as f64);
30     *M.get_unchecked_mut(9)=(log(*M.get_unchecked(8),) as f64);
31     *M.get_unchecked_mut(10)=(*M.get_unchecked(9) / *M.get_unchecked(5) as f64);
32     *M.get_unchecked_mut(11)=(1.0 - *M.get_unchecked(10) as f64);
33     *M.get_unchecked_mut(12)=(log(*M.get_unchecked(11),) as f64);
34     *M.get_unchecked_mut(13)=(*M.get_unchecked(12) * *M.get_unchecked(7) as f64)
           ;
35     *M.get_unchecked_mut(14)=(*M.get_unchecked(13) + *M.get_unchecked(6) as f64)
           ;
36 }
37 pub unsafe fn gompertz_icdf(p:f64,c:f64,loc:f64,scale:f64) -> f64 {
38     *M.get_unchecked_mut(4)=(p as f64);
39     *M.get_unchecked_mut(5)=(c as f64);
40     *M.get_unchecked_mut(6)=(loc as f64);
41     *M.get_unchecked_mut(7)=(scale as f64);
42     _gompertz_icdf();
43     return *M.get_unchecked(14);
44 }
45 unsafe fn _gompertz_sample() {
46     *M.get_unchecked_mut(18)=(uniform(0.0,1.0) as f64);
47     *M.get_unchecked_mut(4)=(*M.get_unchecked(18) as f64);
48     *M.get_unchecked_mut(5)=(*M.get_unchecked(15) as f64);
49     *M.get_unchecked_mut(6)=(*M.get_unchecked(16) as f64);
50     *M.get_unchecked_mut(7)=(*M.get_unchecked(17) as f64);
```

```
51      _gompertz_icdf();
52      *M.get_unchecked_mut(19)=(*M.get_unchecked(14) as f64);
53  }
54  pub unsafe fn gompertz_sample(c:f64,loc:f64,scale:f64) -> f64 {
55      *M.get_unchecked_mut(15)=(c as f64);
56      *M.get_unchecked_mut(16)=(loc as f64);
57      *M.get_unchecked_mut(17)=(scale as f64);
58      _gompertz_sample();
59      return *M.get_unchecked(19);
60  }
61  unsafe fn gompertz_sample_multi(start_addr: usize, n_elements: usize) {
62      for i in 0..n_elements{
63          M[start_addr + i] = gompertz_sample();
64      }
65  }
66  unsafe fn _gaussian() {
67      *M.get_unchecked_mut(23)=(*M.get_unchecked(20) - *M.get_unchecked(21) as f64
            );
68      *M.get_unchecked_mut(24)=(*M.get_unchecked(23) / *M.get_unchecked(22) as f64
            );
69      *M.get_unchecked_mut(25)=((*M.get_unchecked(24)).powf(2.0) as f64);
70      *M.get_unchecked_mut(26)=(-1.0 * *M.get_unchecked(25) as f64);
71      *M.get_unchecked_mut(27)=(exp(*M.get_unchecked(26),) as f64);
72  }
73  pub unsafe fn gaussian(x:f64,mu:f64,sigma:f64) -> f64 {
74      *M.get_unchecked_mut(20)=(x as f64);
75      *M.get_unchecked_mut(21)=(mu as f64);
76      *M.get_unchecked_mut(22)=(sigma as f64);
77      _gaussian();
78      return *M.get_unchecked(27);
79  }
80  unsafe fn _U_icdf() {
81      *M.get_unchecked_mut(33)=(*M.get_unchecked(32) - *M.get_unchecked(31) as f64
            );
82      *M.get_unchecked_mut(34)=(*M.get_unchecked(30) * *M.get_unchecked(33) as f64
            );
83      *M.get_unchecked_mut(35)=(*M.get_unchecked(31) + *M.get_unchecked(34) as f64
            );
84  }
85  pub unsafe fn U_icdf(p:f64,low:f64,high:f64) -> f64 {
```

```
86      *M.get_unchecked_mut(30)=(p as f64);
87      *M.get_unchecked_mut(31)=(low as f64);
88      *M.get_unchecked_mut(32)=(high as f64);
89      _U_icdf();
90      return *M.get_unchecked(35);
91  }
92  unsafe fn _U_sample() {
93      *M.get_unchecked_mut(38)=(uniform(0.0,1.0) as f64);
94      *M.get_unchecked_mut(30)=(*M.get_unchecked(38) as f64);
95      *M.get_unchecked_mut(31)=(*M.get_unchecked(36) as f64);
96      *M.get_unchecked_mut(32)=(*M.get_unchecked(37) as f64);
97      _U_icdf();
98      *M.get_unchecked_mut(39)=(*M.get_unchecked(35) as f64);
99  }
100 pub unsafe fn U_sample(low:f64,high:f64) -> f64 {
101     *M.get_unchecked_mut(36)=(low as f64);
102     *M.get_unchecked_mut(37)=(high as f64);
103     _U_sample();
104     return *M.get_unchecked(39);
105 }
106 unsafe fn U_sample_multi(start_addr: usize, n_elements: usize) {
107     for i in 0..n_elements{
108         M[start_addr + i] = U_sample();
109     }
110 }
111 unsafe fn _P_has_work_icdf() {
112     *M.get_unchecked_mut(20)=(*M.get_unchecked(42) as f64);
113     *M.get_unchecked_mut(21)=(41.56060857 as f64);
114     *M.get_unchecked_mut(22)=(21.78168914 as f64);
115     _gaussian();
116     *M.get_unchecked_mut(43)=(*M.get_unchecked(27) as f64);
117     *M.get_unchecked_mut(44)=((*M.get_unchecked(41)<*M.get_unchecked(43)) as u32
            as f64);
118 }
119 pub unsafe fn P_has_work_icdf(p:f64,age:f64) -> f64 {
120     *M.get_unchecked_mut(41)=(p as f64);
121     *M.get_unchecked_mut(42)=(age as f64);
122     _P_has_work_icdf();
123     return *M.get_unchecked(44);
124 }
```

```
unsafe fn _P_has_work_sample() {
    *M.get_unchecked_mut(46)=(uniform(0.0,1.0) as f64);
    *M.get_unchecked_mut(41)=(*M.get_unchecked(46) as f64);
    *M.get_unchecked_mut(42)=(*M.get_unchecked(45) as f64);
    _P_has_work_icdf();
    *M.get_unchecked_mut(47)=(*M.get_unchecked(44) as f64);
}
pub unsafe fn P_has_work_sample(age:f64) -> f64 {
    *M.get_unchecked_mut(45)=(age as f64);
    _P_has_work_sample();
    return *M.get_unchecked(47);
}
unsafe fn P_has_work_sample_multi(start_addr: usize, n_elements: usize) {
    for i in 0..n_elements{
        M[start_addr + i] = P_has_work_sample();
    }
}
pub unsafe fn D_work_size_sample() -> f64 {
    _D_work_size_sample();
    return *M.get_unchecked(48);
}
unsafe fn _D_work_size_sample() {
    *M.get_unchecked_mut(56)=(bisect(&M,49,55,uniform(0.0, 1.0)) as f64);
    *M.get_unchecked_mut(57)=(*M.get_unchecked(56) - 49.0 as f64);
    ch=(*M.get_unchecked(57) as f64);
    if ch == 0.0 {
        *M.get_unchecked_mut(48)=(uniform(5.0,10.0) as f64);
    } else if ch == 1.0 {
        *M.get_unchecked_mut(48)=(uniform(10.0,25.0) as f64);
    } else if ch == 2.0 {
        *M.get_unchecked_mut(48)=(uniform(25.0,50.0) as f64);
    } else if ch == 3.0 {
        *M.get_unchecked_mut(48)=(uniform(50.0,100.0) as f64);
    } else if ch == 4.0 {
        *M.get_unchecked_mut(48)=(uniform(100.0,200.0) as f64);
    } else if ch == 5.0 {
        *M.get_unchecked_mut(48)=(uniform(200.0,500.0) as f64);
    } else {
        *M.get_unchecked_mut(48)=(uniform(500.0,1000.0) as f64);
    }
```

```
}
unsafe fn D_work_size_sample_multi(start_addr: usize, n_elements: usize) {
    for i in 0..n_elements{
        M[start_addr + i] = D_work_size_sample();
    }
}
pub unsafe fn D_closed_init_sample() -> f64 {
    _D_closed_init_sample();
    return *M.get_unchecked(58);
}
unsafe fn _D_closed_init_sample() {
    *M.get_unchecked_mut(61)=(bisect(&M,59,60,uniform(0.0, 1.0)) as f64);
    *M.get_unchecked_mut(62)=(*M.get_unchecked(61) - 59.0 as f64);
    ch=(*M.get_unchecked(62) as f64);
    *M.get_unchecked_mut(58)=(ch+0.0 as f64);
}
unsafe fn D_closed_init_sample_multi(start_addr: usize, n_elements: usize) {
    for i in 0..n_elements{
        M[start_addr + i] = D_closed_init_sample();
    }
}
pub unsafe fn D_house_size_sample() -> f64 {
    _D_house_size_sample();
    return *M.get_unchecked(63);
}
unsafe fn _D_house_size_sample() {
    *M.get_unchecked_mut(73)=(bisect(&M,64,72,uniform(0.0, 1.0)) as f64);
    *M.get_unchecked_mut(74)=(*M.get_unchecked(73) - 64.0 as f64);
    ch=(*M.get_unchecked(74) as f64);
    *M.get_unchecked_mut(63)=(ch+2.0 as f64);
}
unsafe fn D_house_size_sample_multi(start_addr: usize, n_elements: usize) {
    for i in 0..n_elements{
        M[start_addr + i] = D_house_size_sample();
    }
}
unsafe fn _Work_step() {
}
pub unsafe fn Work_size() -> &'static␣mut␣[f64]␣{
␣␣␣␣&mut␣M[76..10076]
```

```
}
pub unsafe fn Work_closed() -> &'static mut [f64] {
    &mut M[10078..20078]
}
unsafe fn _P_workplace_prob() {
}
pub unsafe fn P_workplace_prob(size:f64,closed:f64) -> f64 {
    *M.get_unchecked_mut(20080)=(size as f64);
    *M.get_unchecked_mut(20081)=(closed as f64);
    _P_workplace_prob();
    return *M.get_unchecked(20080);
}
unsafe fn _P_workplace_sample() {
    let mut probs = [0_f64; (10000) as usize];
    for __i in 0..(10000 as usize) {
        *M.get_unchecked_mut(20082) = __i as f64;
        *M.get_unchecked_mut(20080)=(*M.get_unchecked((76.0) as usize + (*M.
            get_unchecked(20082)) as usize) as f64);
        *M.get_unchecked_mut(20081)=(*M.get_unchecked((10078.0) as usize + (*M.
            get_unchecked(20082)) as usize) as f64);
        _P_workplace_prob();
        *M.get_unchecked_mut(20085)=(*M.get_unchecked(20080) as f64);
        probs[*M.get_unchecked_mut(20082) as usize]=(*M.get_unchecked(20085) as
            f64);
    }
    for __i in  1..10000 { probs[__i] += probs[__i - 1] };
    let s = probs[9999];
    for __i in 0..10000 { probs[__i] /= s };
    *M.get_unchecked_mut(20086)=(uniform(0.0,1.0) as f64);
    *M.get_unchecked_mut(20087)=(bisect(&probs,0,10000,*M.get_unchecked(20086))
        as f64);
    *M.get_unchecked_mut(20088)=(*M.get_unchecked(20087) + 0.0 as f64);
}
pub unsafe fn P_workplace_sample() -> f64 {
    _P_workplace_sample();
    return *M.get_unchecked(20088);
}
pub unsafe fn P_workplace_sample_multi(start_addr:f64,n_samples:f64) -> f64 {
    let mut probs = [0_f64; (10000) as usize];
    for __i in 0..(10000 as usize) {
```

```
241         *M.get_unchecked_mut(20082) = __i as f64;
242         *M.get_unchecked_mut(20080)=(*M.get_unchecked((76.0) as usize + (*M.
                get_unchecked(20082)) as usize) as f64);
243         *M.get_unchecked_mut(20081)=(*M.get_unchecked((10078.0) as usize + (*M.
                get_unchecked(20082)) as usize) as f64);
244         _P_workplace_prob();
245         *M.get_unchecked_mut(20085)=(*M.get_unchecked(20080) as f64);
246         probs[*M.get_unchecked_mut(20082) as usize]=(*M.get_unchecked(20085) as
                f64);
247     }
248     for __i in  1..10000 { probs[__i] += probs[__i - 1] };
249     let s = probs[9999];
250     for __i in 0..10000 { probs[__i] /= s };
251     for i in 0..(n_samples as usize) {
252         *M.get_unchecked_mut((start_addr) as usize + (i) as usize)=(0.0 + bisect
                (&probs, 0, 10000, uniform(0.0, 1.0)) as f64);
253     }
254     return 0.0;
255 }
256 unsafe fn _Household_step() {
257 }
258 pub unsafe fn Household_size() -> &'static␣mut␣[f64]␣{
259 ␣␣␣␣&mut␣M[20090..120090]
260 }
261 pub␣unsafe␣fn␣Household_closed()␣->␣&'static mut [f64] {
262     &mut M[120092..220092]
263 }
264 unsafe fn _P_household_prob() {
265 }
266 pub unsafe fn P_household_prob(size:f64,closed:f64) -> f64 {
267     *M.get_unchecked_mut(220094)=(size as f64);
268     *M.get_unchecked_mut(220095)=(closed as f64);
269     _P_household_prob();
270     return *M.get_unchecked(220094);
271 }
272 unsafe fn _P_household_sample() {
273     let mut probs = [0_f64; (100000) as usize];
274     for __i in 0..(100000 as usize) {
275         *M.get_unchecked_mut(220096) = __i as f64;
276         *M.get_unchecked_mut(220094)=(*M.get_unchecked((20090.0) as usize + (*M.
```

```
            get_unchecked(220096)) as usize) as f64);
277         *M.get_unchecked_mut(220095)=(*M.get_unchecked((120092.0) as usize + (*M
                .get_unchecked(220096)) as usize) as f64);
278         _P_household_prob();
279         *M.get_unchecked_mut(220099)=(*M.get_unchecked(220094) as f64);
280         probs[*M.get_unchecked_mut(220096) as usize]=(*M.get_unchecked(220099)
                as f64);
281     }
282     for __i in  1..100000 { probs[__i] += probs[__i - 1] };
283     let s = probs[99999];
284     for __i in 0..100000 { probs[__i] /= s };
285     *M.get_unchecked_mut(220100)=(uniform(0.0,1.0) as f64);
286     *M.get_unchecked_mut(220101)=(bisect(&probs,0,100000,*M.get_unchecked
            (220100)) as f64);
287     *M.get_unchecked_mut(220102)=(*M.get_unchecked(220101) + 10000.0 as f64);
288 }
289 pub unsafe fn P_household_sample() -> f64 {
290     _P_household_sample();
291     return *M.get_unchecked(220102);
292 }
293 pub unsafe fn P_household_sample_multi(start_addr:f64,n_samples:f64) -> f64 {
294     let mut probs = [0_f64; (100000) as usize];
295     for __i in 0..(100000 as usize) {
296         *M.get_unchecked_mut(220096) = __i as f64;
297         *M.get_unchecked_mut(220094)=(*M.get_unchecked((20090.0) as usize + (*M.
                get_unchecked(220096)) as usize) as f64);
298         *M.get_unchecked_mut(220095)=(*M.get_unchecked((120092.0) as usize + (*M
                .get_unchecked(220096)) as usize) as f64);
299         _P_household_prob();
300         *M.get_unchecked_mut(220099)=(*M.get_unchecked(220094) as f64);
301         probs[*M.get_unchecked_mut(220096) as usize]=(*M.get_unchecked(220099)
                as f64);
302     }
303     for __i in  1..100000 { probs[__i] += probs[__i - 1] };
304     let s = probs[99999];
305     for __i in 0..100000 { probs[__i] /= s };
306     for i in 0..(n_samples as usize) {
307         *M.get_unchecked_mut((start_addr) as usize + (i) as usize)=(10000.0 +
                bisect(&probs, 0, 100000, uniform(0.0, 1.0)) as f64);
308     }
```

```
309     return 0.0;
310 }
311 unsafe fn _Person_step() {
312 }
313 pub unsafe fn Person_age() -> &'static␣mut␣[f64]␣{
314 ␣␣␣␣&mut␣M[220104..1220104]
315 }
316 pub␣unsafe␣fn␣Person_work()␣->␣&'static mut [f64] {
317     &mut M[1220106..2220106]
318 }
319 pub unsafe fn Person_has_work() -> &'static␣mut␣[f64]␣{
320 ␣␣␣␣&mut␣M[2220108..3220108]
321 }
322 pub␣unsafe␣fn␣Person_house()␣->␣&'static mut [f64] {
323     &mut M[3220110..4220110]
324 }
325 pub unsafe fn Person_inf_state() -> &'static␣mut␣[f64]␣{
326 ␣␣␣␣&mut␣M[4220126..5220126]
327 }
328 pub␣unsafe␣fn␣Person_inf_days()␣->␣&'static mut [f64] {
329     &mut M[5220126..6220126]
330 }
331 unsafe fn _Person_schedule() {
332     *M.get_unchecked_mut(4220116)=(*M.get_unchecked((10078.0) as usize + (*M.
        get_unchecked((1220106.0) as usize + (*M.get_unchecked(4220114)) as usize
        )) as usize) * *M.get_unchecked((3220110.0) as usize + (*M.get_unchecked
        (4220114)) as usize) as f64);
333     *M.get_unchecked_mut(4220118)=(1.0 - *M.get_unchecked((10078.0) as usize +
        (*M.get_unchecked((1220106.0) as usize + (*M.get_unchecked(4220114)) as
        usize)) as usize) as f64);
334     *M.get_unchecked_mut(4220119)=((*M.get_unchecked(4220113)>7.0) as u32 as f64
        );
335     *M.get_unchecked_mut(4220120)=(*M.get_unchecked((1220106.0) as usize + (*M.
        get_unchecked(4220114)) as usize) * *M.get_unchecked(4220119) as f64);
336     *M.get_unchecked_mut(4220121)=((*M.get_unchecked(4220113)<=7.0) as u32 as
        f64);
337     *M.get_unchecked_mut(4220122)=(*M.get_unchecked((3220110.0) as usize + (*M.
        get_unchecked(4220114)) as usize) * *M.get_unchecked(4220121) as f64);
338     *M.get_unchecked_mut(4220123)=(*M.get_unchecked(4220120) + *M.get_unchecked
        (4220122) as f64);
```

```
339     *M.get_unchecked_mut(4220124)=(*M.get_unchecked(4220118) * *M.get_unchecked
            (4220123) as f64);
340     *M.get_unchecked_mut(4220125)=(*M.get_unchecked(4220116) + *M.get_unchecked
            (4220124) as f64);
341 }
342 pub unsafe fn Person_schedule(h:f64,i:f64) -> f64 {
343     *M.get_unchecked_mut(4220113)=(h as f64);
344     *M.get_unchecked_mut(4220114)=(i as f64);
345     _Person_schedule();
346     return *M.get_unchecked(4220125);
347 }
348 pub unsafe fn Flu_state_prob_sample() -> f64 {
349     _Flu_state_prob_sample();
350     return *M.get_unchecked(6220127);
351 }
352 unsafe fn _Flu_state_prob_sample() {
353     *M.get_unchecked_mut(6220132)=(bisect(&M,6220128,6220131,uniform(0.0, 1.0))
            as f64);
354     *M.get_unchecked_mut(6220133)=(*M.get_unchecked(6220132) - 6220128.0 as f64)
            ;
355     ch=(*M.get_unchecked(6220133) as f64);
356     *M.get_unchecked_mut(6220127)=(ch+0 as f64);
357 }
358 unsafe fn Flu_state_prob_sample_multi(start_addr: usize, n_elements: usize) {
359     for i in 0..n_elements{
360         M[start_addr + i] = Flu_state_prob_sample();
361     }
362 }
363 unsafe fn _Flu_progress_prob(out: &mut [f64; 4]) {
364     let s = *M.get_unchecked(6220139);
365     let d = *M.get_unchecked(6220140);
366     for i in 0..4 { out[i] = 0.0; }
367     if s == 0.0 {
368         if true {
369             out[1] = -0.014499569695115089;
370             return
371         }
372     }
373     else if s == 1.0 {
374         if d == 14.0 {
```

```
            out[2] = -3.3219280948873626;
            return
        }
    }
    else if s == 2.0 {
        if d == 14.0 {
            out[3] = -39.86313713864835;
            return
        }
    }
}
pub unsafe fn Flu_progress_prob(inf_state:f64,inf_days:f64,out: &mut [f64; 4]) {
    *M.get_unchecked_mut(6220139)=(inf_state as f64);
    *M.get_unchecked_mut(6220140)=(inf_days as f64);
    _Flu_progress_prob(out);
}
unsafe fn _Flu_spread_prob(out: &mut [[f64; 4]; 4]) {
    let sd = *M.get_unchecked((4220126.0) as usize + (*M.get_unchecked(6220154))
        as usize);
    if sd == 1.0 {
        out[0][1] += -0.0002885678659263426;
    }
    else if sd == 2.0 {
        out[0][1] += -0.0007215279174593579;
    }
}
pub unsafe fn Flu_spread_prob(agent_r:f64,agent_d:f64,comp_i:f64,out: &mut [[f64
    ; 4]; 4]) {
    *M.get_unchecked_mut(6220153)=(agent_r as f64);
    *M.get_unchecked_mut(6220154)=(agent_d as f64);
    *M.get_unchecked_mut(6220155)=(comp_i as f64);
    _Flu_spread_prob(out);
}
unsafe fn _Flu_spread_prob_inc(out: &mut [f64; 4]) {
    let sr = *M.get_unchecked((4220126.0) as usize + (*M.get_unchecked(6220153))
        as usize);
    let sd = *M.get_unchecked((4220126.0) as usize + (*M.get_unchecked(6220154))
        as usize);
    if sd == 1.0 {
        if sr == 0.0 {
```

```
411                 out[1] += -0.0002885678659263426;
412             }
413             else if sr == 1.0 {
414             }
415             else if sr == 2.0 {
416             }
417             else if sr == 3.0 {
418             }
419         }
420         else if sd == 2.0 {
421             if sr == 0.0 {
422                 out[1] += -0.0007215279174593579;
423             }
424             else if sr == 1.0 {
425             }
426             else if sr == 2.0 {
427             }
428             else if sr == 3.0 {
429             }
430         }
431 }
432 pub unsafe fn Flu_spread_prob_inc(agent_r:f64,agent_d:f64,comp_i:f64,out: &mut [
        f64; 4]) {
433     *M.get_unchecked_mut(6220153)=(agent_r as f64);
434     *M.get_unchecked_mut(6220154)=(agent_d as f64);
435     *M.get_unchecked_mut(6220155)=(comp_i as f64);
436     _Flu_spread_prob_inc(out);
437 }
438 pub unsafe fn Flu_spread_0_0_branch_sample() -> f64 {
439     _Flu_spread_0_0_branch_sample();
440     return *M.get_unchecked(6220170);
441 }
442 unsafe fn _Flu_spread_0_0_branch_sample() {
443     *M.get_unchecked_mut(6220175)=(bisect(&M,6220171,6220174,uniform(0.0, 1.0))
            as f64);
444     *M.get_unchecked_mut(6220176)=(*M.get_unchecked(6220175) - 6220171.0 as f64)
            ;
445     ch=(*M.get_unchecked(6220176) as f64);
446     *M.get_unchecked_mut(6220170)=(ch+0 as f64);
447 }
```

```rust
unsafe fn Flu_spread_0_0_branch_sample_multi(start_addr: usize, n_elements: usize) {
    for i in 0..n_elements{
        M[start_addr + i] = Flu_spread_0_0_branch_sample();
    }
}
pub unsafe fn Flu_spread_0_1_branch_sample() -> f64 {
    _Flu_spread_0_1_branch_sample();
    return *M.get_unchecked(6220177);
}
unsafe fn _Flu_spread_0_1_branch_sample() {
    *M.get_unchecked_mut(6220182)=(bisect(&M,6220178,6220181,uniform(0.0, 1.0)) as f64);
    *M.get_unchecked_mut(6220183)=(*M.get_unchecked(6220182) - 6220178.0 as f64);
    ch=(*M.get_unchecked(6220183) as f64);
    *M.get_unchecked_mut(6220177)=(ch+0 as f64);
}
unsafe fn Flu_spread_0_1_branch_sample_multi(start_addr: usize, n_elements: usize) {
    for i in 0..n_elements{
        M[start_addr + i] = Flu_spread_0_1_branch_sample();
    }
}
pub unsafe fn Flu_is_susceptible(r:f64) -> f64 {
    *M.get_unchecked_mut(6220153)=(r as f64);
    return *M.get_unchecked((*M.get_unchecked((4220126.0) as usize + (*M.get_unchecked(6220153)) as usize)) as usize + (6220184.0) as usize);
}
pub unsafe fn Flu_is_infectious(d:f64) -> f64 {
    *M.get_unchecked_mut(6220154)=(d as f64);
    return *M.get_unchecked((*M.get_unchecked((4220126.0) as usize + (*M.get_unchecked(6220154)) as usize)) as usize + (6220188.0) as usize);
}
pub unsafe fn infection_spread() {
    let mut probs = Vec::<[[[f64; 4]; 4]; 24]>::with_capacity(110000);
    probs.resize(110000, <[[[f64; 4]; 4]; 24]>::default());
    let mut pr = [[0_f64; 4]; 4];
    let mut pp = [0_f64; 4];
    for p in 0..1000000 {
```

```
482         if Flu_is_infectious(p as f64) < 0.5 { continue }
483         for h in 0..24 {
484             let c = Person_schedule(h as f64, p as f64) as usize;
485             Flu_spread_prob(0.0, p as f64, c as f64, &mut probs[c][h]);
486         }
487     }
488     for p in 0..1000000 {
489         let sd = M[4220126 + p];
490         let dd = M[5220126 + p];
491         Flu_progress_prob(sd, dd, &mut pp);
492         let sd = sd as usize;
493         if Flu_is_susceptible(p as f64) > 0.5 {
494             for h in 0..24 {
495                 let c = Person_schedule(h as f64, p as f64) as usize;
496                 for j in 0..4 {
497                     pp[j] += probs[c as usize][h as usize][sd][j];
498                 }
499             }
500         }
501         let mut sm = 0_f64;
502         for s in 0..4 {
503             if s == sd { continue }
504             pp[s] = 1.0 - (2_f64).powf(pp[s]);
505             sm += pp[s];
506         }
507         pp[sd] = 1.0 - sm;
508         for i in 1..4 { pp[i] += pp[i - 1]; }
509         let new_s = bisect(&pp, 0, 3, uniform(0., 1.));
510         if new_s != sd {
511             M[4220126 + p] = new_s as f64;
512             M[5220126 + p] = 1.0;
513         } else {
514             M[5220126 + p] += 1.0;
515         }
516     }
517 }
518
519 pub unsafe fn step() -> f64 {
520     infection_spread();
521     _Work_step();
```

```
522     _Household_step();
523     _Person_step();
524     *M.get_unchecked_mut(0)=(*M.get_unchecked(0) + 1.0 as f64);
525     return 0.0;
526 }
527 pub unsafe fn init() -> f64 {
528     M.resize(6220192, 0.0);
529     *M.get_unchecked_mut(0)=(0 as f64);
530     *M.get_unchecked_mut(49)=(0.46 as f64);
531     *M.get_unchecked_mut(50)=(0.78 as f64);
532     *M.get_unchecked_mut(51)=(0.905 as f64);
533     *M.get_unchecked_mut(52)=(0.9550000000000001 as f64);
534     *M.get_unchecked_mut(53)=(0.9750000000000001 as f64);
535     *M.get_unchecked_mut(54)=(0.9900000000000001 as f64);
536     *M.get_unchecked_mut(55)=(1.0 as f64);
537     *M.get_unchecked_mut(59)=(1.0 as f64);
538     *M.get_unchecked_mut(60)=(1.0 as f64);
539     *M.get_unchecked_mut(64)=(0.5710764018065374 as f64);
540     *M.get_unchecked_mut(65)=(0.7933022909741141 as f64);
541     *M.get_unchecked_mut(66)=(0.9494891611762494 as f64);
542     *M.get_unchecked_mut(67)=(0.9889991639364591 as f64);
543     *M.get_unchecked_mut(68)=(0.9972477908321032 as f64);
544     *M.get_unchecked_mut(69)=(0.9991389345590265 as f64);
545     *M.get_unchecked_mut(70)=(0.9997059776543017 as f64);
546     *M.get_unchecked_mut(71)=(0.999914993539509 as f64);
547     *M.get_unchecked_mut(72)=(1.0 as f64);
548     for __i in 0..(10000 as usize) {
549         *M.get_unchecked_mut(75) = __i as f64;
550         _D_work_size_sample();
551         *M.get_unchecked_mut(10076)=(*M.get_unchecked(48) as f64);
552         *M.get_unchecked_mut((76.0) as usize + (*M.get_unchecked_mut(75)) as
                usize)=(*M.get_unchecked(10076) as f64);
553         _D_closed_init_sample();
554         *M.get_unchecked_mut(20078)=(*M.get_unchecked(58) as f64);
555         *M.get_unchecked_mut((10078.0) as usize + (*M.get_unchecked_mut(75)) as
                usize)=(*M.get_unchecked(20078) as f64);
556         }
557     for __i in 0..(100000 as usize) {
558         *M.get_unchecked_mut(20089) = __i as f64;
559         _D_house_size_sample();
```

```
560             *M.get_unchecked_mut(120090)=(*M.get_unchecked(63) as f64);
561             *M.get_unchecked_mut((20090.0) as usize + (*M.get_unchecked_mut(20089))
                    as usize)=(*M.get_unchecked(120090) as f64);
562             _D_closed_init_sample();
563             *M.get_unchecked_mut(220092)=(*M.get_unchecked(58) as f64);
564             *M.get_unchecked_mut((120092.0) as usize + (*M.get_unchecked_mut(20089))
                     as usize)=(*M.get_unchecked(220092) as f64);
565             }
566         P_workplace_sample_multi(1220106.0, 1000000.0);
567         P_household_sample_multi(3220110.0, 1000000.0);
568         for __i in 0..(1000000 as usize) {
569             *M.get_unchecked_mut(220103) = __i as f64;
570             *M.get_unchecked_mut(15)=(0.20510679560362763 as f64);
571             *M.get_unchecked_mut(16)=(-1.82765672236963 as f64);
572             *M.get_unchecked_mut(17)=(29.342139434877467 as f64);
573             _gompertz_sample();
574             *M.get_unchecked_mut(1220104)=(*M.get_unchecked(19) as f64);
575             *M.get_unchecked_mut((220104.0) as usize + (*M.get_unchecked_mut(220103)
                    ) as usize)=(*M.get_unchecked(1220104) as f64);
576             *M.get_unchecked_mut(45)=(*M.get_unchecked((220104.0) as usize + (*M.
                    get_unchecked(220103)) as usize) as f64);
577             _P_has_work_sample();
578             *M.get_unchecked_mut(3220108)=(*M.get_unchecked(47) as f64);
579             *M.get_unchecked_mut((2220108.0) as usize + (*M.get_unchecked_mut
                    (220103)) as usize)=(*M.get_unchecked(3220108) as f64);
580             }
581         *M.get_unchecked_mut(6220126)=(1.0 as f64);
582         *M.get_unchecked_mut(6220128)=(1.0 as f64);
583         *M.get_unchecked_mut(6220129)=(1.0 as f64);
584         *M.get_unchecked_mut(6220130)=(1.0 as f64);
585         *M.get_unchecked_mut(6220131)=(1.0 as f64);
586         *M.get_unchecked_mut(6220171)=(0.9998 as f64);
587         *M.get_unchecked_mut(6220172)=(1.0 as f64);
588         *M.get_unchecked_mut(6220173)=(1.0 as f64);
589         *M.get_unchecked_mut(6220174)=(1.0 as f64);
590         *M.get_unchecked_mut(6220178)=(0.9995 as f64);
591         *M.get_unchecked_mut(6220179)=(1.0 as f64);
592         *M.get_unchecked_mut(6220180)=(1.0 as f64);
593         *M.get_unchecked_mut(6220181)=(1.0 as f64);
594         *M.get_unchecked_mut(6220184)=(1 as f64);
```

```
595     *M.get_unchecked_mut(6220185)=(0 as f64);
596     *M.get_unchecked_mut(6220186)=(0 as f64);
597     *M.get_unchecked_mut(6220187)=(0 as f64);
598     *M.get_unchecked_mut(6220188)=(0 as f64);
599     *M.get_unchecked_mut(6220189)=(1 as f64);
600     *M.get_unchecked_mut(6220190)=(1 as f64);
601     *M.get_unchecked_mut(6220191)=(0 as f64);
602     Flu_state_prob_sample_multi(4220126, 1000000);
603     return 0.0;
```

ДОДАТОК Б. Визначення формальної граматики мови CTrace надане розширеною формою Бекуса-Наура

```
PLUS ::= "+"
MINUS ::= "-"
MUL ::= "*"
DIV ::= "/"
POW ::= "^"
JOIN ::= "@"
EQUAL ::= "=="
NOTEQUAL ::= "<>"
LESSER ::= "<"
GREATER ::= ">"
LESSEREQUAL ::= "<="
GREATEREQUAL ::= ">="
ASSIGN ::= "="
LPAREN ::= "("
RPAREN ::= ")"
LCURL ::= "{"
RCURL ::= "}"
LBRACK ::= "["
RBRACK ::= "]"
DOT ::= "."
SAMPLEFROM ::= "~"
SPECIFIER_SYMBOL ::= "$"
RARROW ::= "->"
LARROW ::= "<-"
VBAR ::= "|"
DIGIT ::= [0-9]
NUMBER ::= (PLUS|MINUS)?DIGIT*(".".DIGIT*)?
LETTER ::= [a-zA-Z]
ID ::= LETTER (DIGIT|LETTER)*
program ::= instruction+
instruction ::= declaration | param_sampling
declaration ::= param_declaration | distribution_declaration |
    function_declaration | struct_declaration
function_declaration ::= ID LPAREN function_declaration_arguments RPAREN ASSIGN
    expression SEMICOLON
function_declaration_arguments ::= ID (COMMA ID)*
param_declaration ::= ID ASSIGN expression SEMICOLON
```

```
distribution_declaration ::= distribution_cont_declaration |
    distribution_disc_declaration
distribution_disc_declaration ::= ID LCURL (value_probability_list|
    interval_probability_list) RCURL
value_probability_list ::= value_probability+
value_probability ::= PROB LPAREN expression RPAREN EQUAL expression SEMICOLON
interval_probability_list ::= interval_probability+
interval_probability ::= PROB LPAREN ((expression COMMA expression) | ((LPAREN |
    LBRACK) expression COMMA expression (RPAREN | RBRACK))) RPAREN EQUAL
    expression SEMICOLON
distribution_cont_declaration ::= (ID LPAREN function_declaration_arguments
    RPAREN LCURL function_declaration RCURL) | (ID LCURL function_declaration
    RCURL)
param_sampling ::= (param_tuple SAMPLEFROM ID SEMICOLON) | (param_tuple
    SAMPLEFROM ID LPAREN function_call_arguments RPAREN SEMICOLON)
struct_declaration ::= specifiers? ID LBRACK NUMBER RBRACK LCURL struct_fields
    RCURL
struct_fields ::= struct_field+
struct_field ::= param_sampling | param_declaration | function_declaration |
    transition
infection ::= ID LCURL infection_states spread_rule+ progress_rule+ RCURL
infection_states ::= ID (COMMA ID)* SEMICOLON
spread_rule ::= PROB LPAREN ID RARROW ID VBAR ID RPAREN ASSIGN expression
    SEMICOLON
progress_rule ::= PROB LPAREN ID RARROW ID (COMMA NUMBER)? RPAREN ASSIGN
    expression SEMICOLON
expression ::= term |
    (expression PLUS expression) |
    (expression MINUS expression) |
    (expression MUL expression) |
    (expression DIV expression) |
    (expression POW expression) |
    (expression JOIN expression) |
    (expression EQUAL expression) |
    (expression NOTEQUAL expression) |
    (expression LESSER expression) |
    (expression GREATER expression) |
    (expression LESSEREQUAL expression) |
    (expression GREATEREQUAL expression) |
    (LPAREN expression RPAREN) |
```

```
    (ID LPAREN function_call_arguments RPAREN)
function_call_arguments ::= expression (COMMA expression)*
term ::= NUMBER | ID | (ID DOT ID)
specifiers ::= specifier+
specifier ::= SPECIFIER_SYMBOL specifier_type
specifier_type ::= SPECIFIER_COMPARTMENT | SPECIFIER_AGENT
transition ::= (ID RARROW expression) | (expression LARROW ID)
```

ДОДАТОК В. Опис епідеміологічної моделі розповсюдження коронавірусу SARS-CoV-2 серед населення Польщі у період з початку вересня 2020 року до кінця листопада 2020 року

```
gompertz(c, loc, scale) {
    icdf(p) = ln(1-ln(1-p)/c)*scale+loc;
}
gaussian(x, mu, sigma) =
    exp(-1 * ((x - mu) / sigma) ^ 2);
U(low, high) {
    icdf(p) = low + p * (high - low);
}

P_has_work(age) {
    icdf(p) = p < gaussian(
        age,
        40.28473393,
        19.23584194
    );
}

D_work_size {
    P(5, 10) = 0.46;
    P(10, 25) = 0.32;
    P(25, 50) = 0.125;
    P(50, 100) = 0.05;
    P(100, 200) = 0.02;
    P(200, 500) = 0.015;
    P(500, 1000) = 0.01;
}

D_house_size {
    P(2 ) = 0.421262;
    P(3 ) = 0.292581;
    P(4 ) = 0.206986;
    P(5 ) = 0.057490;
    P(6 ) = 0.014664;
    P(7 ) = 0.004384;
    P(8 ) = 0.001531;
```

```
    P(9 ) = 0.000639;
    P(10) = 0.000209;
}
$compartment
Work[1150] {
    size ~ D_work_size;
}
P_workplace(Work) {
    prob(w) = size;
}
$compartment
Household[12500] {
    size ~ D_house_size;
}
P_household(Household) {
    prob(h) = size;
}

$agent
Person[38000] {
    age ~ gompertz(
        0.24669550087983944,
        -1.7802394687678404,
        30.049961335205445
    );
    work ~ P_workplace;
    has_work ~ P_has_work(age);
    house ~ P_household;

    schedule(h) = work;
}

Flu {
    S 38000, E 6, I 0, R 0;

    P(S -> E | I) = 0.009951400344083403;
    P(S -> E) = 0.0010502822181501664;

    P(E -> I) = 0.0009116555869307609;

```

```
76      P(I -> R) = 0.1328123197247606;
77  }
```

ДОДАТОК Г. Список публікацій здобувача за темою дисертацій

Публікації у фахових виданнях, включених до переліку наукових фахових видань України з присвоєнням категорії «Б»:

1. Сарнацький В. В., Баклан І. В. МЕТОДИ ТА ЗАСОБИ МОДЕЛЮВАННЯ РОЗПОВСЮДЖЕННЯ ІНФЕКЦІЙНИХ ЗАХВОРЮВАНЬ. Вчені записки Таврійського національного університету імені В. І. Вернадського. т. 33. No 2. р. 2022. с 100-107;
2. Сарнацький В. В., Баклан І. В. ВПЛИВ СОЦІОДЕМОГРАФІЧНОЇ ГЕТЕРОГЕННОСТІ НА ОПТИМАЛЬНУ СТРАТЕГІЮ ВПРОВАДЖЕННЯ КАРАНТИННИХ ЗАХОДІВ. Вчені записки Таврійського національного університету імені В. І. Вернадського. т. 32. No 4. р. 2021. с 150-155;
3. Sarnatskyi V., Baklan I. CTraceEnv: A platform for development and analysis of agent-based epidemiological models using CTrace language. Вчені записки Таврійського національного університету імені В. І. Вернадського. т. 34. No 2. р. 2023. с. 100-107.

Публікації у наукових виданнях, включених до наукометричної бази Scopus:

4. Sarnatskyi V., Baklan I. On Efficient Single-Core Execution of Agent-Based Epidemiological Models with Contact-Tracing Transmission. Proceedings of The Sixth International Workshop on Computer Modeling and Intelligent Systems (CMIS 2023), vol.3392, 2023. – с. 23-36;
5. Sarnatskyi V., Baklan I. CTrace: Language for Definition of Epidemiological Models with Contact-Tracing Transmission. Lecture Notes on Data Engineering and Communications Technologies, vol.149 – Springer, Cham, 2023. – С. 426-448.

Публікації у матеріалах наукових конференцій:

6. Сарнацький, В. В. Мовний засіб опису агентних епідеміологічних моделей / Сарнацький Владислав Віталійович, Баклан Ігор Всеволодович // Інженерія програмного забезпечення і передові інформаційні технології (SoftTech-2022)

: матеріали II та III Всеукраїнських науково-практичних конференцій молодих вчених та студентів, присвячених 125-й річниці КПІ ім. Ігоря Сікорського (22–26 травня та 23-25 листопада 2022 р., Київ). – Київ : КПІ ім. Ігоря Сікорського, ІПІ ФІОТ, 2022. – С. 85-89. – Бібліогр.: 17 назв.