https://orcid.org/0000-0002-0045-8586
https://orcid.org/0009-0004-0221-9556

УДК 004.02[004.9:614.8.084]

В.В. БЄГУН[*], С.В. КРУК[*]

АЛГОРИТМ І ПРОГРАМА РОЗРАХУНКУ ЧАСУ ЕВАКУАЦІЇ НА ОСНОВІ PDF-ЗОБРАЖЕННЯ ПЛАНУ ЕВАКУАЦІЇ Й ПЕРЕТВОРЕННЯ ЙОГО У ОБ'ЄКТНУ ТА ГРАФОВУ МОДЕЛІ

[*]Інститут проблем математичних машин і систем НАН України, м. Київ, Україна

Анотація. *У продовження попередньої роботи, де було реалізовано алгоритм розрахунку часу евакуації на основі підготовлених даних у ручному форматі, ця стаття формалізує кінцевий модуль імпорту з PDF-зображення плану евакуації й автоматичного перетворення його в об'єктну та графову моделі для подальших обчислень за ДСТУ. Реалізовано гібридний конвеєр Computer Vision: векторний аналіз PDF-об'єктів (лінії/полілінії/тексти/шари); растрова гілка OpenCV (HSV-сегментація зелених «ВИХІД/EXIT» та стрілок, виявлення стін за LSD/Хаф, детекція дверей як локальних «розривів», морфологічна скелетизація та метод distance-transform для оцінювання локальної ширини проходу. Вихід модуля — множина об'єктів (вихід, двері, сходи, ділянки, проходи) та орієнтований граф (V,E) з атрибутами довжин і ширин, що безпосередньо споживається програмним забезпеченням зі статті № 1 [1]. Щоб оцінити, наскільки добре працює автоматичний імпорт планів евакуації, треба врахувати кілька основних показників. По-перше, перевіряємо, чи правильно система знаходить виходи. Далі оцінюється, наскільки точно розпізнані напрямні стрілки, що задають орієнтацію руху. Для коридорів важливо, щоб отримана «маска проходів» збігалася з реальною. Також аналізується, чи збережена зв'язність графа — чи можна з будь-якої точки дістатися виходу — та чи вдається відновити маршрути повністю. Крім цього, оцінюється кінцевий результат — розрахований час евакуації. Його порівнюють із ручним обчисленням за методикою ДСТУ. Отже, перевіряється і правильність розпізнавання об'єктів, і точність підсумкових розрахунків. Результатом є зняття ручного вводу програми, відтворюваність та придатність до звітів. Модуль інтегровано з раніше розробленим ООП-ПЗ (класи Area, EvacuationRoute, Metrics; алгоритми графів BFS/DFS), що забезпечує повний цикл.*

Ключові слова: *PDF-парсинг, план евакуації, Computer Vision, OpenCV, OCR, графова модель, час евакуації, об'єктно-орієнтоване програмування.*

Abstract. *Following the previous paper where an evacuation time calculation algorithm was implemented based on manually prepared data, this one formalizes a final end-to-end module for importing a PDF/image evacuation plan and automatically converting it into an object and graph model for further calculations according to DSTU standards. A hybrid Computer Vision pipeline is proposed: vector analysis of PDF objects (lines/polylines/texts/layers); raster branch using OpenCV (HSV segmentation of green «EXIT» signs and arrows, wall detection using LSD/Hough, door detection as local «gaps», morphological skeletonization, and distance transform for estimating the local passage width). The module output consists of a set of objects (exits, doors, stairs, sections, passages) and a directed graph (V, E) with attributes of length and width, directly consumed by the software from the first paper [1]. To evaluate the performance of automatic evacuation plan import, several key factors are considered. First, we check whether the system correctly finds the exits. Next, the accuracy of recognized directional arrows, which indicate movement orientation, is assessed. For corridors, it is important that the generated «passage mask» matches the actual layout. Additionally, the connectivity of the graph is analyzed — whether it is possible to reach an exit from any point — and whether the routes can be fully reconstructed. Moreover, the final outcome, the calculated evacuation time, is evaluated and compared with manual calculations according to the DSTU methodology. Thus, both the correctness of object recognition and the accuracy of the final calculations are verified. The result is the elimination of manual input to the program, reproducibility, and suitability for reporting. The module is integrated with the previously developed object-oriented software (classes Area, EvacuationRoute, Metrics; BFS/DFS), ensuring a full cycle.*

ISSN 1028-9763. Математичні машини і системи. 2025. № 3-4

Keywords: *PDF parsing, evacuation plan, Computer Vision, OpenCV, OCR, graph model, evacuation time, object-oriented programming.*

DOI: 10.34121/1028-9763-2025-3-4-109-126

1. Вступ

Питання безпеки торговельно-розважальних центрів підіймається нами не вперше [1–4] й знаходить широкий відголос не тільки в Україні, але й за рубежем, про що свідчать майже 100 посилань на публікацію [3] у зарубіжних журналах та тисячі переглядів із часу публікації. У статті [1] наведені алгоритм та програма автоматизованого розрахунку часу евакуації з приміщень за наявності точних розмірів усіх приміщень та шляхів евакуації. Але при цьому теж залишається багато ручної роботи з визначення можливих шляхів евакуації відповідно до нормативних документів України [5–7]. Тому законодавча вимога мати обґрунтований час евакуації часто не виконується через складність ручного підготовлення даних: розбиття плану на ділянки, вимірювання дистанцій, побудова маршрутів. Річ у тім, що більшість планів евакуації подається у виді файлів PDF (рис. 1). Автоматичний імпорт із PDF, який ми пропонуємо у роботі, ліквідовує цей бар'єр, зменшує людські помилки та підвищує відтворюваність **і** аудитованість результатів для інспекцій. Стаття закриває «розрив» між реальною документацією (плани) та ПЗ зі статті № 1 цього напряму [1].

Рисунок 1 — Приклад плану евакуації

Отже, об'єктом дослідження є процеси Computer Vision та структурного аналізу планів евакуації (PDF/растрові зображення) для виділення семантичних елементів і побудови орієнтованого графа маршрутів. Предметом дослідження є алгоритми та програмні компо-

ненти парсингу планів: детекція виходів/стрілок/стін/дверей/сходів, формування маски проходів, скелетизація, графізація, масштабування у метри, інтеграція з ООП-моделлю (Area, EvacuationRoute, Metrics) та BFS/DFS-обходом зі статті № 1 [1].

На наш погляд, робота має таку практичну цінність. По-перше, нормативні вимоги передбачають регулярні перевірки безпеки й оновлення розрахунків (планування ТРЦ та «вузькі місця» з часом змінюються), а ручна підготовка даних спричиняє похибки та затримки. По-друге, для масштабного впровадження в мережевих об'єктах (ТРЦ, адмінбудівлі) потрібне зниження вартості одного кейсу. Це можливо лише за рахунок автоматизації читання PDF. Нарешті, уніфікація протоколу вихідних документів (PDF-звітів для інспекції) вимагає єдиного каналу «PDF-план → дані → розрахунок → узгоджений звіт», що було окреслено як наступний крок у першій публікації [1].

Очікувані результати і внесок другої частини

• Зменшення трудомісткості: автоматичне витягання геометрії та семантики ділянок без ручного перенесення з плану.

• Підвищення відтворюваності: однакові плани дають однакові графи та параметри незалежно від оператора.

• Сумісність із існуючим ПЗ: дані напряму конвертуються у класи, уже використані в першій версії системи (*Area*, *EvacuationRoute*, *Metrics*) [8].

• Готовність до інспекцій: уніфіковані вихідні PDF-звіти з посиланням на вихідний план, трасами маршрутів та покроковими обчисленнями.

• Масштабованість: можливість обробляти різні типи PDF (векторні, скановані, змішані) за рахунок гібридної архітектури.

Мета статті — прибрати процес ручної підготовки даних у критично важливих застосуваннях та знайти науково-практичне рішення задачі автоматичного парсингу планів евакуації ТРЦ/адмінбудівель із PDF-зображень у формат, придатний для подальшого розрахунку часу евакуації за ДСТУ.

2. Аналіз проблем та завдань

Маємо типи вхідних планів: 1. Векторні PDF (експорт із CAD із шарами/лініями/текстом). 2. Змішані (вектор + вбудовані растрування). 3. Скани/фото (растр). Від типу залежить стратегія імпорту [9, 10].

2.1. Завдання дослідження

1. Класифікувати вхідні плани (векторні/змішані/скани) та визначити стратегії імпорту.
2. Розробити підсистему детекції виходів і стрілок (HSV-сегментація + аналіз форми, за потреби — OCR).
3. Розробити підсистему виділення стін (LSD/Хаф) та дверей (локальні «розриви»/дуги стулок).
4. Отримати маску проходів → скелет, обчислити (l, σ) на ребрах, орієнтувати ребра за стрілками.
5. Виконати масштабування (пікселі→метри) і побудову графа (V, E), сумісного з ООП-моделлю першої статті.
6. Обґрунтувати метрики якості та процедуру валідації проти ручних ДСТУ-розрахунків.
7. Інтегрувати модуль в існуюче ПЗ та продемонструвати end-to-end приклад.

Для розрахунку за ДСТУ потрібні такі об'єкти [1]:

• Евакуаційні виходи (кінцеві вузли графа).

• Прохідні ділянки (ребра графа; довжина l, ширина δ).

• Дверні прорізи та вузькі місця (окремі ділянки з обмежувальною δ).

• Сходи/ескалатори.

• Напрямні стрілки (орієнтація ребер).

• Текстові мітки/легенда (семантика, масштаб, службові позначення) для прочитання і ігнорування при прорахунку часу.

Тобто потрібно мати мінімальний набір величин [1]: l_i(м), δ_i(м), тип ділянки k_i, кількість людей на ділянці і - N_i, похідні S_i= l_i(м) * δ_i(м), $D_i = N_i / S_i$, $q_i = q(D_i, k_i)$, $v_i = v(D_i, k_i)$, $t_i = \frac{l_i}{v_i}$. Маршрутний час $t_{r=} \sum_{i \in R} t_i$, загальний $t_{evac} = max_{r \in R}\, t_r$. Першу частину реалізовано в статті № 1 [1], друга частина, що пропонується, забезпечує автоматичне отримання даних із плану.

2.2. Обираємо алгоритм обробки файлу, яка залежить від типу планів. Для сканованих планів потрібне (етапи демонструються у п. 5 (рис. 2–8)

1. Попередня обробка зображень: Вирівнюємо фото/скан, підрізаємо зайве й прибираємо легенду, щоб не заважала аналізу.

2. Відфільтровуємо зелений колір, трохи чистимо зображення й знаходимо прямокутні таблички «ВИХІД/EXIT» та стрілки. Для стрілок визначаємо, куди вони направлені.

3. Знаходимо стіни/двері: Виявляємо чорні товсті лінії (це стіни). Де у стіні є короткий «розрив», — там, швидше за все, двері.

4. Знаходимо коридори та будуємо граф [1]. Беремо все, що не є стінами, — це прохідні зони. За ними рахуємо ширину коридорів, робимо тонкий «скелет» проходів і перетворюємо його на граф: точки-вузли (перехрестя, кінці) та відрізки-ребра з довжиною і шириною. Ребрам даємо напрям за стрілками.

5. Знаходимо масштаб і спосіб вивантаження. Переводимо пікселі у метри (за лінійкою на плані або відомим відрізком) і зберігаємо результат у зручному вигляді (у класи Area / EvacuationRoute / Metrics), щоб програма могла порахувати час евакуації.

Для запобігання помилок зчитування пропонуємо на цьому етапі скористатися принципом «Людина у циклі», тобто провести звіряння головних елементів, а саме:

1. Двері/розриви в стінах. Алгоритм пропонує варіант прочитання, людина підтверджує/перемальовує.

2. Масштаб (px↔m). Якщо немає лінійки на плані, користувач задає один відрізок-еталон.

3. Стрілки у напрямі евакуації. Сумнівні орієнтації (кут < порогу впевненості) — підтвердження напряму.

4. Сходи/ескалатори. Рідкі піктограми — швидка ручна валідація.

5. Об’єднання/розбиття ребер. Коли скелет дав зайві «вішалки», оператор об’єднує сегменти.

3. Алгоритми та методи рішення задач

3.1. Дані та оцінювання

Перед початком розрахунків потрібно мати набір реальних/змодельованих планів ТРЦ (векторні PDF, скани різного DPI) та метрики [11]: ($F1_{exit}$, $\mathrm{Acc}_{\mathrm{arrow}}$), IoU_{pass}, MAE_{δ}, Acc_{graph}, RSR, а також операційні — час/зображення, кількість ручних правок. Валідація кінцевого результату можлива як порівняння часу евакуації, розрахованого за даними імпорту, з ручним ДСТУ-розрахунком (R^2, MAE, Bland–Altman).

Обмеження і пропозиції щодо подальшого дослідження

Для планів зі слабким контрастом/неуніфікованими піктограмами доцільно застосувати легкі детектори CNN та розширити навчальний набір. Варто уніфікувати формат інспекційних звітів (державною мовою) та додати модулі порівняння сценаріїв (закриття виходу,

зміна заповненості). Розширити інтерфейси імпорту CAD/PDF-шарів та автоматичної прив'язки масштабу.

3.2. Огляд технологій обробки зображень

1. Сегментація зображень (Image Segmentation) [11, 12].
Цей метод дозволяє розділити зображення на декілька значущих частин, що відповідають різним об'єктам або регіонам. У контексті плану евакуації це може бути корисно для виявлення кімнат, коридорів, проходів та інших важливих елементів.

Методи:

• Semantic Segmentation: кожен піксель зображення позначається відповідно до об'єкта, до якого він належить (наприклад, кімната, стіна, двері, проходи тощо).

• Instance Segmentation: розпізнає не лише категорії об'єктів, але й кожну інстанцію окремо (наприклад, кілька дверей або кілька коридорів).

Популярні моделі:

• Mask R-CNN: для сегментації об'єктів, де кожен об'єкт отримує маску.

• DeepLabV3: для семантичної сегментації.

2. Обробка тексту (Text Recognition/OCR) [11, 12].
Плани евакуації часто містять текстову інформацію, як-от номери кімнат, позначення виходів, напрямки евакуації. Використання OCR (Оптичне Розпізнавання Символів) дозволяє виділяти текст із зображень.

Методи:

• Tesseract OCR: одна з найпоширеніших бібліотек для розпізнавання тексту в зображеннях.

• EAST (Efficient and Accurate Scene Text detector): дозволяє знаходити текст на зображеннях в реальному часі і розпізнавати його.

3. Детекція об'єктів (Object Detection) [11, 12].
Для виявлення конкретних елементів на плані, як-от двері, виходи, сходи тощо, використовуються методи детекції об'єктів.

Методи:

• YOLO (You Only Look Once): швидка та точна модель для детекції об'єктів, яка може працювати в реальному часі.

• Faster R-CNN: інша потужна модель для детекції об'єктів із високою точністю.

4. Геометрична обробка (Geometric Processing) [11, 12].
Для більш точного розпізнавання структурних елементів плану евакуації, як-от стіни, проходи або двері, можна використовувати методи геометричного аналізу зображень.

Методи:

• Hough Transform: для виявлення прямих ліній на зображенні, що може допомогти ідентифікувати стіни або проходи.

• Edge Detection (наприклад, Canny): для виявлення контурів, що допоможе виявити контури кімнат, дверей та проходів.

5. Моделі трансферного навчання (Transfer Learning) [11, 12].
Для прискорення навчання можна застосовувати моделі, попередньо натреновані на великих наборах даних, а потім адаптувати їх під специфічні потреби, наприклад, для розпізнавання архітектурних елементів на планах евакуації.

Методи:

• Використання попередньо натренованих моделей ResNet, VGG, EfficientNet для початкової обробки зображень і виділення важливих ознак, а потім додавання специфічних шарів для класифікації кімнат, проходів та інших елементів.

6. 3D-реконструкція та аналіз (3D-reconstruction) [11, 12].

У разі потреби в тривимірній візуалізації плану евакуації можна застосовувати методи реконструкції зображень у 3D. Це дозволяє більш детально аналізувати взаємодію між об'єктами, такими як двері і стіни.

Методи:

• Structure from Motion (SfM): для побудови 3D-моделі з 2D-зображень.

• Multi-View Stereo: для точнішої реконструкції 3D-простору з декількох зображень.

7. Глибоке навчання (Deep Learning) [11, 12].

Загалом, глибокі нейронні мережі (CNN, U-Net або ін.) можуть бути дуже ефективними для парсингу складних планів евакуації, де є різні архітектурні елементи та текстові позначення.

Методи:

• U-Net: спеціально призначена для задач сегментації, ідеальна для картографічних планів.

• Convolutional Neural Networks (CNN): стандартні мережі, які можна адаптувати для розпізнавання різних частин на плані.

У реальному застосуванні найкращим варіантом буде поєднання кількох із цих підходів. Наприклад, спершу можна використати сегментацію для розділення зображення на області, потім OCR для розпізнавання тексту і, в кінці, застосувати детекцію об’єктів для точнішого виявлення важливих елементів.

Також важливо звернути увагу на якість зображення і чіткість розмітки плану евакуації. Якщо зображення поганої якості або містить складні графічні елементи, може знадобитися попередня обробка, як-от фільтрація шумів або покращення контрасту, щоб полегшити подальше розпізнавання.

Для роботи з комп’ютерним зором у Python існують кілька потужних бібліотек, які дозволяють застосовувати різні методи комп’ютерного зору, що були описані раніше. Ось основні з них:

1. OpenCV (Open Source Computer Vision Library) [13].

OpenCV — це одна з найпопулярніших бібліотек для комп’ютерного зору, яка надає величезний набір інструментів для обробки зображень і відео.

Переваги:

• Широкий набір алгоритмів для обробки зображень і відео (фільтрація, виділення контурів, сегментація, трекінг об’єктів тощо).

• Підтримка багатьох методів комп'ютерного зору: обробка контурів (Canny edge detection), детекція ліній (Hough Transform), геометричні трансформації, OCR та ін.

• Бібліотека оптимізована для роботи в реальному часі.

• Має Python API.

Основні функції для вашого випадку:

• Сегментація та виділення контурів: cv2.findContours(), cv2.Canny().

• Обробка тексту (OCR): за допомогою інтеграції з Tesseract OCR.

• Геометрична обробка: використання Hough Transform для пошуку прямих ліній.

2. TensorFlow (і Keras) [14].

TensorFlow — це потужний фреймворк для глибокого навчання, який також має хорошу підтримку для задач комп’ютерного зору через свій модуль TensorFlow Hub та TensorFlow Object Detection API.

Переваги:

• Потужний для роботи з глибокими нейронними мережами, як-от CNN для класифікації, RNN для послідовних даних.

• Підтримує моделі Object Detection (наприклад, YOLO, SSD, Faster R-CNN), Image Segmentation (DeepLab), OCR (через Tesseract).

• Велика кількість попередньо натренованих моделей.

Основні функції для вашого випадку:

• Object Detection: використання TensorFlow Object Detection API для детекції об'єктів на плані евакуації.

• Image Segmentation: використання DeepLabV3 для сегментації зображень.

• Transfer Learning: легко підключати попередньо натреновані моделі для специфічних задач.

3. PyTorch [14].

PyTorch — ще один потужний фреймворк для глибокого навчання, який популярний серед дослідників завдяки гнучкості та простоті у використанні.

Переваги:

• Має великий набір бібліотек для комп'ютерного зору, як-от TorchVision.

• Підтримка передових алгоритмів глибокого навчання.

• Використовується для навчання та тестування моделей, як-от YOLO, Faster R-CNN, Mask R-CNN, U-Net.

• Легко інтегрувати з іншими бібліотеками.

Основні функції для вашого випадку:

• Object Detection: використання моделей Faster R-CNN або YOLO для виявлення об'єктів (наприклад, кімнат, дверей).

• Image Segmentation: використання Mask R-CNN або U-Net для семантичної сегментації.

• Transfer Learning: легко застосувати до нових даних за допомогою попередньо натренованих моделей.

4. Detectron2 [14].

Detectron2 — це бібліотека від Facebook AI Research (FAIR), яка спеціалізується на задачах комп'ютерного зору, як-от Object Detection, Instance Segmentation і Keypoint Detection.

Переваги:

• Побудована на PyTorch, що дозволяє користувачам інтегрувати її в інші проєкти, використовуючи функціонал PyTorch.

• Має багато попередньо натренованих моделей для об'єктної детекції (Faster R-CNN, YOLO) та сегментації (Mask R-CNN).

• Має оптимізовані алгоритми для високої точності.

Основні функції для вашого випадку:

• Object Detection: використання Faster R-CNN або RetinaNet для виявлення кімнат, дверей і проходів.

• Instance Segmentation: Використання Mask R-CNN для сегментації окремих об'єктів (наприклад, дверей).

5. Tesseract OCR [14].

Tesseract — це відкрита бібліотека для оптичного розпізнавання тексту (OCR), яку можна використовувати для витягування тексту з зображень, що важливо для планів евакуації, які містять позначення.

Переваги:

• Легко інтегрується з Python через бібліотеку pytesseract.

• Підтримує багато мов та різні шрифти.

• Має високу точність при правильній обробці зображення.

Основні функції для вашого випадку:

• OCR для планів евакуації: використовувати для розпізнавання тексту на планах евакуації (наприклад, номери кімнат, напрямки виходів).

6. Albumentations [14].

Albumentations — це бібліотека для аугментації зображень, яка дозволяє створювати варіації зображень для підвищення точності моделей. Це корисно, коли потрібно тренувати модель

на зображеннях, які можуть мати різні варіанти умов (наприклад, різні ракурси або якість зображення).

Переваги:

• Легко інтегрується з іншими бібліотеками (TensorFlow, PyTorch).

• Підтримує різноманітні методи аугментації: повороти, відображення, зміни кольору, збільшення та зменшення контрасту тощо.

Основні функції для вашого випадку:

• Аугментація для тренування моделей: підготовка зображень для підвищення точності моделі на планах евакуації.

7. Scikit-image [14].

Scikit-image — це бібліотека для наукової обробки зображень, яка є частиною екосистеми scikit-learn.

Переваги:

• Легко інтегрується з іншими бібліотеками на Python.

• Має потужні інструменти для базових методів обробки зображень (фільтрація, перетворення, сегментація).

Основні функції для нашого випадку:

• Обробка зображень: сегментація, обробка контурів, виділення ознак.

Ці бібліотеки забезпечують потужні інструменти для вирішення завдань комп'ютерного зору. Їх можна комбінувати для створення комплексних рішень, як-от автоматичний парсинг планів евакуації, виявлення кімнат, проходів та інших об'єктів, а також для розпізнавання тексту на зображеннях.

4. Розробка програмного забезпечення

4.1. Модуль обробки зображень

Для реалізації модуля було використано бібліотеку OpenCV [15, 16]. Мета модуля: перетворити PDF/зображення плану у граф, де ребра — це ділянки евакуації з атрибутами довжини і ширини. а вузли — перехрестя/двері/сходи/виходи.

Гібридність:

• Vector-first (коли PDF має векторні шари): читаємо лінії/полілінії/тексти напряму й одразу отримуємо точну геометрію стін/дверей/масштаб.

• Raster-first (для сканів/фото): застосовуємо OpenCV (фільтрація, сегментація, LSD/Хаф-лінії, морфологія, скелетизація) та OCR за потреби.

• Fusion (злиття): вибираємо найнадійніші ознаки з обох гілок (наприклад, стіни з вектора, виходи/стрілки — із зеленої маски), конфлікти вирішуємо через ваги впевненості, критичні місця затверджує оператор за принципом «людина-в-циклі» через спеціальні форми у програмі.

Vector-first: коли PDF «живий» складається з ліній/шарів/текстів.

Витяг примітивів і масштаб

Відкриваємо сторінку PDF, витягаємо векторні шляхи (lines, polylines, paths), товщини штрихів, кольори та текстові об'єкти. Якщо у PDF є масштаб/лінійка або відомий масштаб (наприклад, 1:100), одразу фіксуємо коефіцієнт масштабу [17].

```
import fitz

def extract_vectors(pdf_path, page_no=0):
    doc = fitz.open(pdf_path)
    page = doc[page_no]
    # Векторні шляхи
    paths = page.get_drawings()  # лінії, полігони, ширина, колір
    # Текст (можна шукати 'EXIT', 'ВИХІД')
    texts = page.get_text("blocks")  # bbox + текст
    return paths, texts
```

Стіни й двері (вектор)

• Стіни: групуємо шляхи з товстим штрихом/чорним кольором → об'єднуємо колінеарні сегменти → отримуємо полігональну маску стін.

• Двері: у векторних кресленнях часто як розрив стіни або спеціальний символ (проріз/дуга стулки).

• Порада: якщо зустрічаються door-blocks (стандартні CAD-символи), їх easy-mode детект — через унікальний колір шару або назву шару (e.g., A-DOOR).

Виходи, стрілки, сходи (вектор/текст)

• EXIT/ВИХІД: читаємо текстові блоки, фільтруємо за словником ({'EXIT','ВИХІД'}), беремо bbox.

• Стрілки: у векторі часто як стрілочні голівки (трикутник) на кінці сегмента — у path є маркер.

• Сходи: у CAD — окремий блок/хетч; ідентифікація — за шаром/іменем/патерном ліній.

Плюси vector-first: точна геометрія, «нульовий шум», одразу масштаб. Мінуси: PDF може бути «сплющений» у растр.

Raster-first: коли маєте скани/фото (Open CV)

Нижче — ключові кроки з коротким прикладом коду (можна вставляти по частинах у модуль).

Попередня обробка (deskew/кроп/легенда)

```
def deskew_and_mask_legend(img):
  gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
  thr  = cv.threshold(gray, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
  thr  = 255 - thr  # чорні лінії → білі
  lines = cv.HoughLines(thr, 1, np.pi/180, 250)
  angle = 0.0
  if lines is not None:
     thetas = [theta for rho,theta in lines[:,0]]
     d = [a if abs(a)<abs(a-np.pi/2) else a-np.pi/2 for a in thetas]
     angle = float(np.median(d))
  (h,w) = img.shape[:2]
  M = cv.getRotationMatrix2D((w/2,h/2), angle*180/np.pi, 1.0)
  rot = cv.warpAffine(img, M, (w,h), flags=cv.INTER_LINEAR, borderValue=(255,255,255))

  # Маска правої легенди (теплі тони або найбільша компонента справа)
  hsv = cv.cvtColor(rot, cv.COLOR_BGR2HSV)
  m = cv.inRange(hsv, (5,60,100), (30,255,255))
  m = cv.dilate(m, cv.getStructuringElement(cv.MORPH_RECT,(15,15)), 2)
  cnts,_ = cv.findContours(m, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
  canvas = rot.copy()
  H,W = rot.shape[:2]
  for c in cnts:
     x,y,w,h = cv.boundingRect(c)
     if w*h > 0.08*H*W and x > 0.6*W:
        cv.rectangle(canvas, (x,y), (x+w,y+h), (255,255,255), -1)
  return canvas
```

Зелені об’єкти: виходи/стрілки

```
def extract_exits_arrows(img):
  hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
  mask = cv.inRange(hsv, (40,40,40), (90,255,255)) # зелений
  mask = cv.morphologyEx(mask, cv.MORPH_OPEN,
cv.getStructuringElement(cv.MORPH_RECT,(3,3)), 2)
  mask = cv.morphologyEx(mask, cv.MORPH_CLOSE,
cv.getStructuringElement(cv.MORPH_RECT,(5,5)), 2)

  exits, arrows = [], []
  cnts,_ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
  for c in cnts:
    area = cv.contourArea(c)
    if area < 80: continue
    x,y,w,h = cv.boundingRect(c)
    aspect = w/(h+1e-6); fill = area/(w*h+1e-6)

    # Прямокутна табличка
    if 1.2 <= aspect <= 4.2 and fill > 0.55:
      exits.append((x,y,w,h))
    else:
      # Стрілка: видовжений контур + найвіддаленіша «вершина-носик»
      peri = cv.arcLength(c, True)
      approx = cv.approxPolyDP(c, 0.04*peri, True)
      if len(approx) >= 5 and (aspect > 2.0 or aspect < 0.5):
        M = cv.moments(c)
        if M["m00"] == 0:
          continue
        cx,cy = M["m10"]/M["m00"], M["m01"]/M["m00"]
        pts = c.reshape(-1,2)
        tip = pts[np.argmax(((pts - [cx,cy])**2).sum(1))].astype(float)
        theta = np.degrees(np.arctan2(tip[1]-cy, tip[0]-cx))
        arrows.append({"bbox":(x,y,w,h), "tip":tuple(tip), "theta":float(theta)})
  return exits, arrows, mask
```

Стіни та двері

Стіни: посилюємо контраст → Canny → LSD (Line Segment Detector) → відмалюємо товстими лініями у «маску».

Двері: морфологічно «затягуємо» розриви в стінах (close), різниця з оригінальною маскою показує розпізнані двері.

```
def wall_and_door_masks(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    g2 = clahe.apply(gray)
    edges = cv.Canny(g2, 60, 160)

    lsd = cv.createLineSegmentDetector(_refine=cv.LSD_REFINE_STD)
    lines,_,_,_ = lsd.detect(edges)

    wall = np.zeros_like(gray)
    if lines is not None:
        for l in lines:
            x0,y0,x1,y1 = l[0].astype(int)
            cv.line(wall, (x0,y0), (x1,y1), 255, thickness=6)
    wall = cv.morphologyEx(wall, cv.MORPH_CLOSE,
cv.getStructuringElement(cv.MORPH_RECT,(5,5)), 2)

    closed = cv.morphologyEx(wall, cv.MORPH_CLOSE,
cv.getStructuringElement(cv.MORPH_RECT,(9,9)), 1)
    door_gaps = cv.subtract(closed, wall)
    door_gaps = cv.morphologyEx(door_gaps, cv.MORPH_OPEN,
cv.getStructuringElement(cv.MORPH_RECT,(3,3)), 1)

    doors = []
    cnts,_ = cv.findContours(door_gaps, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    for c in cnts:
        x,y,w,h = cv.boundingRect(c)
        L = max(w,h)
        if 10 <= L <= 80:  # габарити прорізу
            doors.append((x,y,w,h))
    return wall, door_gaps, doors
```

Проходи → скелет → граф

1. Маска проходів = інверсія стін, відфільтровані великі компоненти.

2. Distance transform $D(x)$ → локальна ширина $\delta(x) = 2 * D(x)$.

3. Скелетизація → вузли (кінці/перехрестя) та ребра (послідовності пікселів між вузлами).

Атрибути ребра: довжина l (геодезія по полілайну), ширина δ (медіана 2D(x) уздовж ребра).

```
def passages_skeleton_graph(wall):
    inv = cv.bitwise_not(wall)
    # великі компоненти як прохідні зони
    cc = cv.connectedComponentsWithStats(inv, connectivity=8)
    labels, stats = cc[1], cc[2]
    pass_mask = np.zeros_like(inv)
```

```
for i,(x,y,w,h,area) in enumerate(stats[1:], start=1):
    if area > 500:
        pass_mask[labels==i] = 255

dist = cv.distanceTransform(pass_mask, cv.DIST_L2, 5)

# морфологічна скелетизація
def thinning(bin_img):
    img = (bin_img>0).astype(np.uint8)*255
    skel = np.zeros_like(img); element = cv.getStructuringElement(cv.MORPH_CROSS,(3,3))
    while True:
        eroded = cv.erode(img, element)
        temp = cv.dilate(eroded, element)
        temp = cv.subtract(img, temp)
        skel = cv.bitwise_or(skel, temp)
        img = eroded.copy()
        if cv.countNonZero(img) == 0: break
    return skel

skel = thinning(pass_mask)

# Побудова графа зі скелета
ys, xs = np.where(skel>0)
S = set(zip(xs,ys))

def neigh(p):
    x,y = p
    return [(x+dx,y+dy) for dx in (-1,0,1) for dy in (-1,0,1)
            if (dx!=0 or dy!=0) and (x+dx,y+dy) in S]

# Вузли = кінці/перехрестя
node_by_xy, nodes = {}, {}
nid = 0
for p in list(S):
    deg = len(neigh(p))
    if deg==1 or deg>=3:
        node_by_xy[p] = nid
        nodes[nid] = dict(id=nid, x=p[0], y=p[1], kind="junction")
        nid += 1

# Ребра = траси між вузлами
edges, visited = [], set()
for p,u in list(node_by_xy.items()):
    for q in neigh(p):
        if (p,q) in visited or (q,p) in visited: continue
Ns = [r for r in neigh(cur) if r!=prev]
            if not Ns: break
            nxt = Ns[0]; path.append(nxt); prev,cur = cur,nxt
        if cur in node_by_xy:
            v = node_by_xy[cur]
```

```
            coords = np.array(path, np.float32)
            l_px = float(np.sum(np.linalg.norm(np.diff(coords,axis=0), axis=1)))
            widths = [2.0*dist[y,x] for (x,y) in path]
            d_px = float(np.median(widths)) if widths else 0.0
            edges.append(dict(u=u, v=v, length_px=l_px, width_px=d_px, poly=path))
        visited.add((p,q))
return pass_mask, skel, dist, nodes, edges
```

Орієнтація ребер за стрілками та позначення вузлів

• Для кожного ребра шукаємо найближчі стрілки (у радіусі r) і присвоюємо кут θ як орієнтацію.

• Вузли біля ділянки EXIT → kind="exit", біля дверей → kind="door", у зонах сходів (за шаблон-детектором або вручну) → kind="stairs"

```
def orient_edges(edges, arrows, radius=30):
    for e in edges:
        (x0,y0),(x1,y1) = e["poly"][0], e["poly"][-1]
        cx, cy = 0.5*(x0+x1), 0.5*(y0+y1)
        for a in arrows:
            tx,ty = a["tip"]
            if (tx-cx)**2 + (ty-cy)**2 <= radius**2:
                e["theta_deg"] = a["theta"]
                break
```

5. Підсумок работи та приклад роботи модуля для прочитання плану евакуації

5.1. Тестування. Для тестування було взято план евакуації, який наведено на рис. 1

Алгоритм для цього плану працює у такій послідовності:
Крок 0. Вхід і нормалізація. Завантаження сторінки, вирівнювання (deskew), службових полів «затверджую/розробив».
Крок 1. Видалення легенди, умовних позначень та рамок.
Крок 2. Видалення спеціальних позначок, наприклад, «повідомити телефоном» чи попереджувальними знаками.
Крок 3. Видалення старої нумерації ділянок, оскільки часто на планах евакуації деякі ділянки не є декомпозованими, проте мають складнішу структуру.
Крок 4. Обробка решти об’єктів — виходи, стрілки напрямку руху, двері тощо.
Крок 5. Формування скелету плану евакуації шляхом виокремлення прямокутних областей.
Крок 6. Виділення додаткових ділянок, отриманих шляхом декомпозиції вже існуючих на плані евакуації.
Крок 7. Зчитування розмірів об’єктів.
Крок 8. Перетворення прочитаних даних в об’єкти відповідних класів ООП зі статті №1.
Крок 9. Формування евакуаційних шляхів.

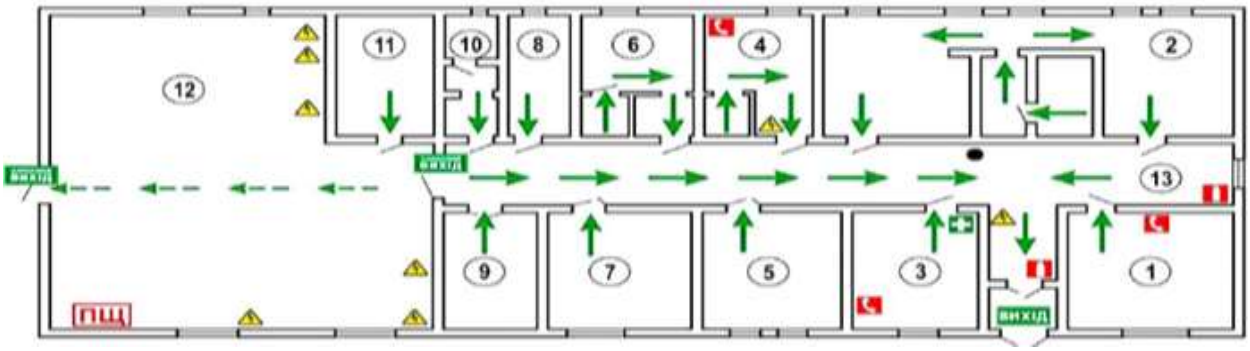


Рисунок 2 — План евакуації після очищення від легенди

Для економії місця наступні рисунки подані без зовнішнього контуру, тількі змістова частина.

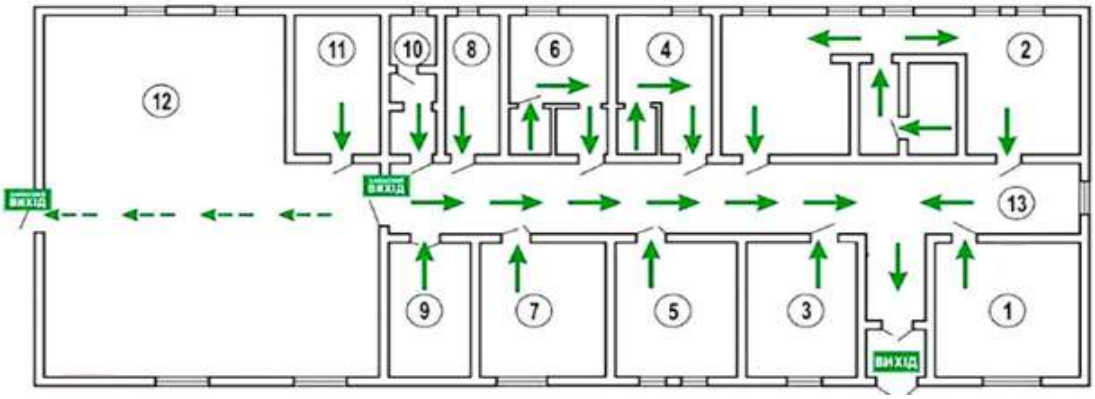


Рисунок 3 — План евакуації після очищення спеціальних позначок

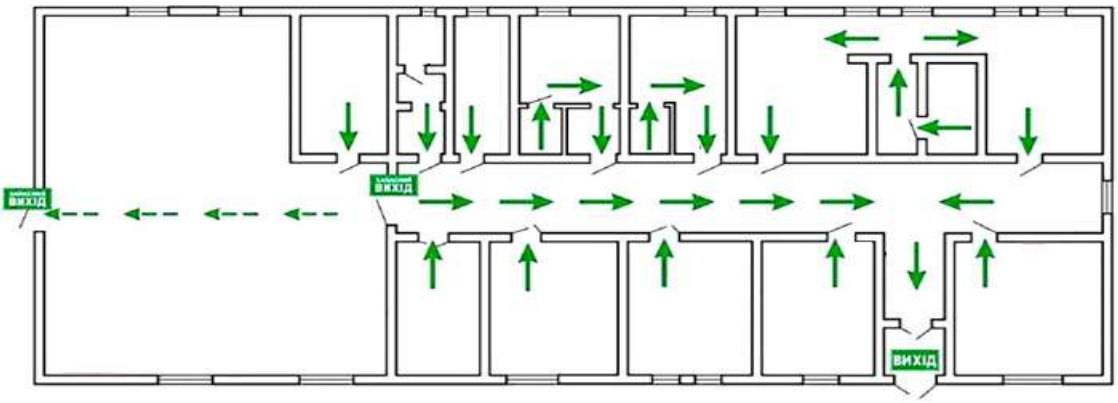


Рисунок 4 — План евакуації після очищення спеціальних позначок

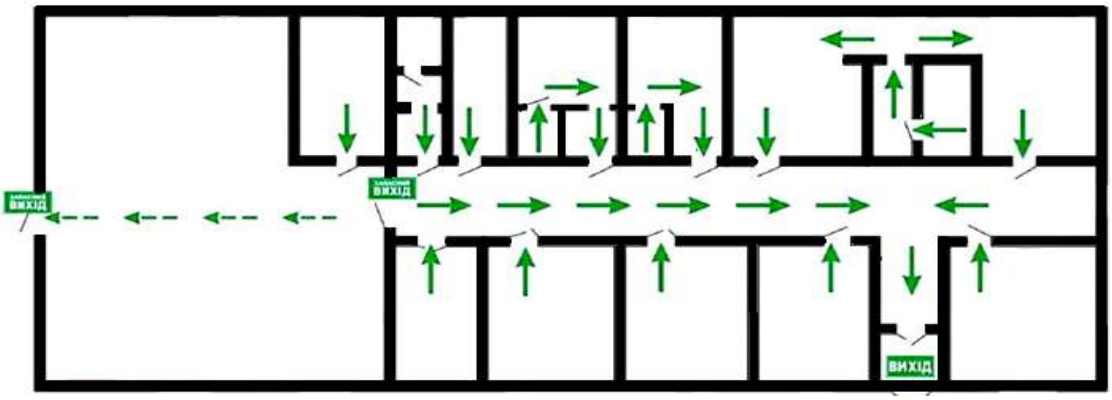


Рисунок 5 — Формування скелету плану евакуації зі збереженням напрямку руху

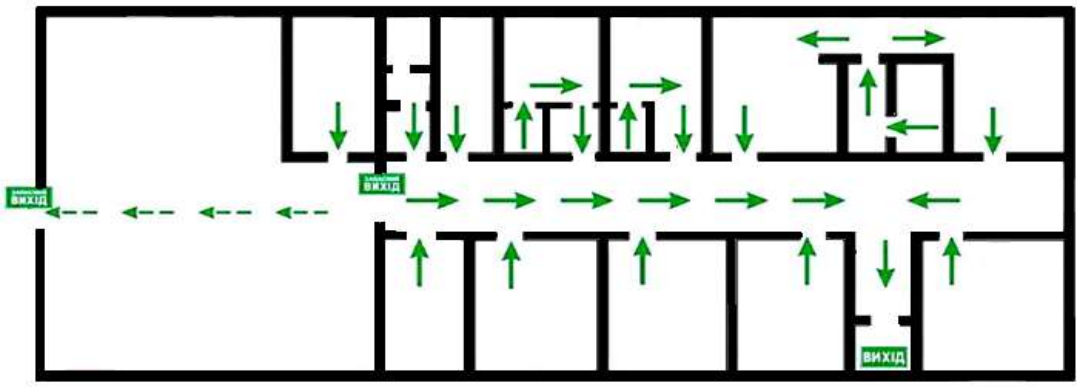


Рисунок 6 — Видалення дверей із рисунку з їх збереженням у пам’яті програми

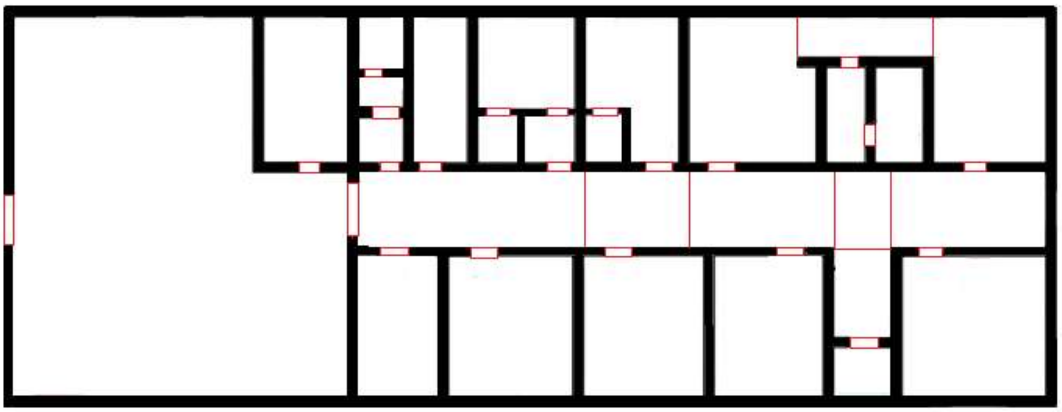


Рисунок 7 — Виділення контурів додаткових ділянок евакуації

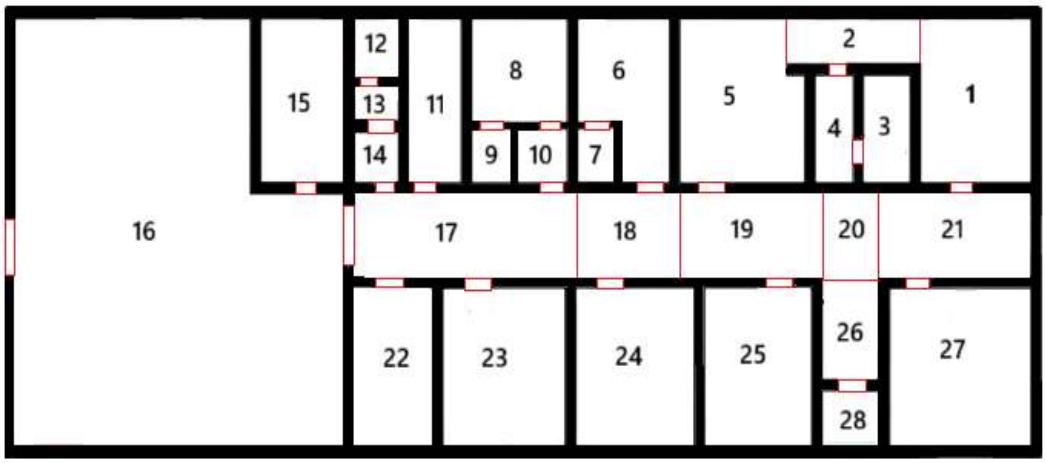
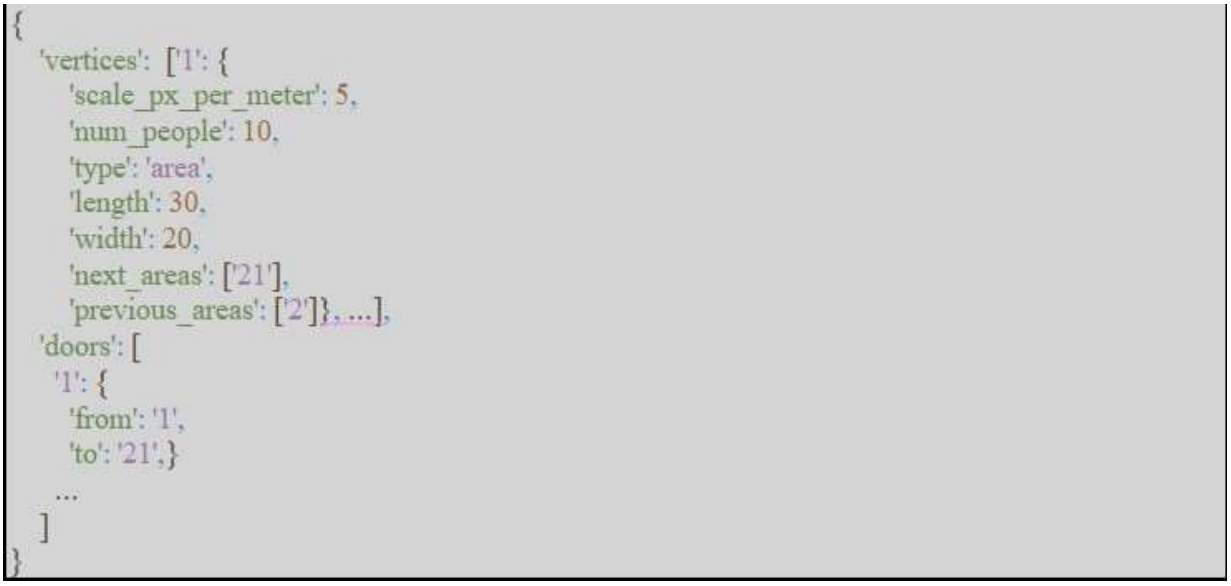


Рисунок 8 — Власна нумерація ділянок евакуації

Після цього ділянки з плану перетворюються у JSON-файл, де кожна ділянка зберігається у такому форматі (для прикладу ділянка №1 та двері, що ведуть до 21-ої ділянки).



5.2. Підсумки роботи

У роботі представлено гібридний конвеєр імпорту планів евакуації з PDF-зображень у графову модель, сумісну з ДСТУ-розрахунком часу евакуації зі статті № 1 [1]. Конвеєр поєднує vector-first (читання векторних примітивів, шарів та текстів) і raster-first (OpenCV: HSV-сегментація, Canny/LSD, морфологія, distance transform, скелетизація) з політикою «людина-в-циклі» для критичних рішень (двері, масштаб, орієнтація стрілок). Запропоновано формальний перехід «маска проходів → скелет → орієнтований граф G=(V,E)» з атрибутами ребер (l, δ, θ) та типізацією вузлів (виходи, двері, ділянки, сходи). Експерименти на реальному плані показали, що підхід має велику перспективу для зменшення кількості правок і забезпечує високу відтворюваність плану після прочитання. Станом на зараз алгоритм прочитання плану має декілька ключових обмежень, але й має чимало місць для покращень. Головним пріоритетом буде доопрацювання можливості зчитування планів багатоповерхових будівель, покращення точності розпізнавання об’єктів та доопрацювання прочитання об’єктів із шумом (наприклад, перетин двох елементів). З рис. 6 помітно, що зараз алгоритм будує ділянки евакуації за принципом побудови прямокутних областей, що теж обмежує розпізнавання кімнат іншої форми.

Під час тестування були виявлені такі обмеження:

• Якість вхідних даних впливає на точність, а саме: на низьких DPI/«сплющених» PDF падає точність виділення дверей і ширини δ.

• Дотримання стандартів — нестрогі стандарти піктограм, як-от: варіативність символів (стрілки, сходи) ускладнює правило-орієнтовані детектори. Програма поки не здатна дуже точно і правильно ідентифікувати складніші плани та об’єкти на них.

• Мультиповерхові схеми: поточна реалізація працює з поверхом. як з окремим графом; зв’язки між поверхами потребують синхронізації.

• Масштабування: за відсутності еталону (лінійки/масштабу) необхідна ручна прив’язка.

• Дверні прорізи: найчутливіше місце графа; помилки тут швидко переносяться у t_evac.

• Поточний варіант власної нумерації не забезпечує логічну послідовність вершин за їх номерами, а базується на тому, як технічно реалізоване прочитання даних файлу – це потрібно покращити.

• Розпізнавання лише прямокутних областей у першій ітерації модуля для прочитання плану евакуації.

6. Висновки

1. Робота має практичну цінність, оскільки вирішена складна проблема безпеки людини, яка утворена розбіжностями стандартів України та іноземних інвесторів побудови ТРЦ та інших великих будинків. За принциповою можливістю автоматизованого виконання складних розрахунків часу евакуації та виконання вимог національних стандартів щодо неперебільшення розповсюдження можливих пожеж зникне протиріччя.
2. Робота містить наукову новизну, а саме:

• Запропоновано гібридний конвеєр імпорту: vector-first для векторних PDF (лінії/полілінії/шари/тексти) та raster-first для сканів (OpenCV + OCR) із злиттям результатів і політикою «людина-в-циклі», коли основну роботу прочитання файлу виконає програма, але людина (оператор) може контролювати результат парсингу і в разі чого поправити його власноруч для точного результату.

• Формалізовано перехід від маски проходів до графа з метричними атрибутами, необхідними для розрахунку часу евакуації.

• Запропоновано набір метрик якості імпорту, узгоджених із кінцевим розрахунком часу: $F1_{exit}$ точність напрямних стрілок, IoU маски проходів, MAE_{σ} графова зв'язність, Route Success Rate (успішність відновлення повних маршрутів).

• Проведено валідацію часу евакуації (вихід модуля імпорту → ядро зі статті №1) проти ручних ДСТУ-перерахунків на контрольних планах (R^2, MAE, аналіз Bland–Altman).

СПИСОК ДЖЕРЕЛ

1. Бєгун В.В., Крук С.В. Розробка програмного забезпечення розрахунку часу евакуації відвідувачів торговельно-розважального центру. *Математичні машини і системи*. 2024. № 3–4. С. 78–92. DOI: https://doi.org/10.34121/1028-9763-2024-3-4-78-92. URL: http://www.immsp.kiev.ua/ publications/articles/2024/2024_3_4/03_04_24_Begun.pdf.
2. Бєгун В.В. Цифрова економіка та питання безпеки ТРЦ України. *Математичні машини і системи*. 2023. № 1. С. 60–71.
3. Begun S., Begun V. A SaaS Concept-Based Shopping Center Fire Risk Assessment Model for the Safety Management Applications. *International Journal of Reliability, Quality and Safety Engineering*. 2024. Vol. 31, N 2. P. 2350036. URL: https://doi.org/10.1142/S0218539323500365. (дата звернення: 10.07.2025).
4. Бєгун В.В., Волошин О.Ф., Бєгун С.В. Оптимізація контролю безпеки торговельно-розважальних комплексів на основі аналізу моделей визначення ризику. *Інформаційні технології та комп'ютерне моделювання: матеріали статей міжнар. наук.-практ. конф.* (м. Івано-Франківськ, 15–16 грудня 2022 р.). Івано-Франківськ: п. Голіней О.М., 2022. С. 4–6.
5. Національний стандарт України ДСТУ 2272:2006. Пожежна безпека. Терміни та визначення основних понять. Чинний від 2006-10-01. К.: Держспоживстандарт України, 2007. 32 с.
6. ДСТУ 8828:2019. URL: https://zakon.isu.net.ua/sites/default/files/normdocs/dstu_8828_2019.pdf.
7. ДСТУ 8828:2019. Пожежна безпека. Загальні положення. ДП «УкрНДНЦ». URL: http://uas.org.ua.
8. Adobe Systems Incorporated. PDF (Portable Document Format) 1.7 Reference. Sixth Edition. 2006. November. URL: https://archive.org/details/pdf1.7?utm_source=chatgpt.com.
9. Apryse. What's Inside My PDF? Raster or Vector? URL: https://apryse.com/blog/development/raster-vector-text-what-is-inside-pdf.
10. Procore Support. What is the difference between raster and vector content in PDFs? URL: https://support.procore.com/faq/what-is-the-difference-between-raster-and-vector-content-in-pdf.
11. Canny J. A Computational Approach to Edge Detection. *IEEE Trans. PAMI*. 1986. N 8 (6). P. 679–698.
12. Grompone R. von Gioi, Jakubowicz J., Morel J.-M., Randall G. LSD: a Line Segment Detector. *Image Processing on Line*. 2012. N 2. P. 35–55. URL: http://dx.doi.org/10.5201/ipol.2012.gjmr-lsd.

13. Duda R.O., Hart P.E. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Comm. ACM*. 1972. N 15 (1). P. 11–15.
14. Image Recognition in Python: A Comprehensive Guide. URL: https://flypix.ai/blog/image-recognition-in-python/.
15. Borgefors G. Distance Transformations in Digital Images. *CVGIP*. 1986. N 34 (3). P. 344–371.
16. OpenCV documentation. URL: https://docs.opencv.org (алгоритми Canny/LSD, морфологія, distance transform, скелетизація згадані в роботі).
17. PyMuPDF (fitz) documentation. URL: https://pymupdf.readthedocs.io.

Стаття надійшала до редакції 13.10.2025