DOI: https://doi.org/10.36910/6775-2524-0560-2026-63-09
УДК: 004.738.5
Берник Анна Олегівна, здобувачка освіти
Гришанович Тетяна Олександрівна, к.ф.-м.н., доцент
https://orcid.org/0000-0002-3595-6964
Павленко Юлія Степанівна, ст. викладач,
https://orcid.org/0000-0002-4065-045X
Глинчук Людмила Ярославівна, к.ф.-м.н., доцент
https://orcid.org/0000-0002-8943-9604
Волинський національний університет імені Лесі Українки, м. Луцьк, Україна

ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ РЕАЛІЗАЦІЇ АЛГОРИТМУ ПОШУКУ МАРШРУТІВ ІЗ ПЕРЕСАДКАМИ

Берник А.О., Гришанович Т.О., Павленко Ю.С., Глинчук Л.Я. Проєктування та розробка веборієнтованої інформаційної системи для реалізації алгоритму пошуку маршрутів із пересадками. У статті розглянуто задачу пошуку оптимальних маршрутів у транспортних мережах із урахуванням пересадок, часових обмежень та особливостей розкладів руху. Актуальність дослідження зумовлена зростанням складності транспортних систем і потребою у швидкому та зручному підборі маршрутів у веборієнтованих інформаційних сервісах. Запропоновано програмну реалізацію алгоритму пошуку маршрутів, який дозволяє знаходити як прямі сполучення, так і маршрути з однією пересадкою з урахуванням мінімального та максимального часу очікування між рейсами. Особливу увагу приділено коректній обробці часових параметрів, зокрема переходу через межі доби, що забезпечує формування реалістичних варіантів поїздок. Розроблено веборієнтовану інформаційну систему для продажу та обліку залізничних квитків, побудовану на основі архітектурного шаблону MVT із використанням фреймворку Django. Система реалізує повний цикл взаємодії з користувачем: пошук маршрутів, перевірку наявності місць, бронювання та оплату квитків, а також підтримує багаторівневу систему доступу. Проведене тестування підтвердило коректність роботи алгоритму як для прямих маршрутів, так і для маршрутів із пересадками, включаючи граничні випадки. Отримані результати демонструють ефективність запропонованого підходу та доцільність його використання у сучасних транспортних інформаційних системах.
Ключові слова: маршрут, пошук маршрутів, веборієнтована система, алгоритми маршрутизації, інформаційна система, проєктування інформаційних систем, оптимізація маршрутів.

Bernyk A., Hryshanovych T., Pavlenko Y., Hlynchuk L. Design and Implementation of a Web-Based Information System for Multi-Leg Route Search Algorithms. The article addresses the problem of finding optimal routes in transportation networks, taking into account transfers, time constraints, and timetable specifics. The relevance of the study is driven by the increasing complexity of transportation systems and the need for fast and convenient route selection in web-oriented information services. A software implementation of a route search algorithm are proposed, enabling the identification of both direct connections and routes with a single transfer, considering minimum and maximum waiting times between trips. Special attention is given to the correct handling of temporal parameters, particularly transitions across day boundaries, which ensures the generation of realistic travel options. A web-oriented information system for railway ticket sales and management has been developed, based on the MVT architectural pattern using the Django framework. The system supports the full cycle of user interaction: route search, seat availability checking, booking, and ticket payment, as well as a multi-level access control system. The conducted testing confirmed the correctness of the algorithm for both direct routes and routes with transfers, including edge cases. The obtained results demonstrate the effectiveness of the proposed approach and the feasibility of its application in modern transportation information systems.
Keywords: route, route planning, web-based information system, routing algorithms, information systems design, route optimization.

Постановка проблеми. Сучасні транспортні системи характеризуються значною складністю структури маршрутів, великою кількістю рейсів та динамічністю розкладів. У таких умовах забезпечення ефективного пошуку та бронювання залізничних квитків стає важливою задачею як для операторів перевезень, так і для користувачів. Традиційні підходи, що базуються на використанні касових систем або спрощених інформаційних сервісів, часто не забезпечують достатньої гнучкості, швидкодії та зручності взаємодії, особливо в умовах зростання обсягів даних і кількості запитів.

Існуючі програмні рішення нерідко обмежуються пошуком прямих маршрутів або використовують недостатньо ефективні алгоритми обробки розкладів, що ускладнює підбір оптимальних варіантів перевезення з урахуванням пересадок. Крім того, проблема ускладнюється необхідністю врахування часових обмежень, таких як мінімальний та максимальний час пересадки, а також забезпечення коректності часових обчислень при переході через межі доби. Водночас

важливим аспектом є забезпечення цілісності даних, зокрема уникнення ситуацій подвійного бронювання місць, що вимагає узгодженості операцій у базі даних.

Окремої уваги потребує інтеграція алгоритмічних рішень із веборієнтованими інформаційними системами, які повинні забезпечувати зручний інтерфейс, багаторівневу систему доступу та підтримку різних ролей користувачів. При цьому постає задача поєднання ефективних методів пошуку маршрутів із вимогами до продуктивності, масштабованості та надійності такого програмного забезпечення.

Таким чином, актуальною є проблема розробки автоматизованої інформаційної системи для продажу та обліку залізничних квитків, яка поєднує ефективний алгоритм пошуку маршрутів (як прямих, так і з пересадками) з урахуванням часових обмежень, забезпечує цілісність даних і водночас відповідає вимогам сучасних вебтехнологій щодо зручності використання та продуктивності. Вирішення цієї проблеми дозволить підвищити ефективність обслуговування користувачів та оптимізувати процес планування поїздок у транспортних системах.

Метою роботи є розробка веборієнтованої інформаційної системи, спрямованої на підвищення ефективності пошуку маршрутів між пунктами сполучення. Запропонований програмний продукт покликаний замінити або доповнити традиційні касові рішення, забезпечуючи користувачам можливість самостійно здійснювати пошук маршрутів, перевіряти наявність вільних місць, а також виконувати бронювання й оплату квитків через вебінтерфейс.

Функціонал системи повинен охоплювати впровадження процедур реєстрації та автентифікації користувачів із розмежуванням рівнів доступу. Слід передбачити поділ на ролі: звичайні користувачі та працівники, серед яких — адміністратори, касири, провідники й керівники станцій. Система має забезпечувати можливість пошуку рейсів за визначеними критеріями (пункт відправлення, пункт призначення, дата), надавати актуальні відомості про ціни квитків і наявність місць, а також підтримувати бронювання та оформлення замовлень. Додатково повинна бути передбачена підтримка різних платіжних інструментів.

Адміністративна частина системи повинна надавати інструменти для швидкого коригування розкладу руху поїздів і керування обліковими записами користувачів. До нефункціональних характеристик належать гарантування надійного збереження даних, висока продуктивність під час обробки запитів, а також зручний і зрозумілий інтерфейс, пристосований до роботи на різних пристроях.

Окремий акцент слід зробити на забезпеченні узгодженості даних, що є важливим для уникнення ситуацій подвійного бронювання одного місця. Також система має використовувати удосконалений підхід до пошуку маршрутів, який враховує як прямі рейси, так і варіанти з пересадками з урахуванням допустимого часу очікування. Це сприятиме підвищенню якості та ефективності підбору оптимальних маршрутів перевезення.

Аналіз останніх досліджень і публікацій. Проблема пошуку оптимальних маршрутів у транспортних мережах із розкладом (timetable-based routing) [1] є однією з ключових задач у галузях транспортної логістики, теорії графів та інформаційних систем. Вона включає врахування часових обмежень, пересадок, вартості та інших критеріїв оптимізації.

Базові алгоритми маршрутизації ґрунтуються на графових моделях, зокрема на алгоритмах пошуку найкоротшого шляху, таких як алгоритм Дейкстри. У роботі [3] показано, що модифіковані варіанти алгоритму Дейкстри можуть ефективно враховувати часові обмеження та розклади руху поїздів. Подібні підходи також застосовуються у задачах планування руху в логістичних системах, де необхідно враховувати часові вікна та обмеження безпеки [9]. Разом з тим, класичні графові алгоритми мають обмеження при роботі з транспортними розкладами, оскільки часові залежності роблять ваги ребер динамічними.

Одним із найбільш відомих сучасних підходів є алгоритм RAPTOR (Round-Based Public Transit Routing), запропонований у роботі Delling et al. [5]. Цей алгоритм працює за принципом ітерацій (раундів), де кожен раунд відповідає кількості пересадок, та знаходить множину Парето-оптимальних маршрутів за кількома критеріями (час прибуття, кількість пересадок). Перевагою RAPTOR є відсутність необхідності попередньої обробки графа та висока швидкодія в динамічних умовах.

Подальші дослідження у цьому напрямку розвивають такий підхід, зокрема, розширюють пошук для врахування вартості поїздки та тарифних моделей [7], оптимізують продуктивності через

методи раннього відсікання (early pruning), що дозволяє зменшити час обчислень до 57% [10], адаптують пошук до мультимодальних транспортних систем та враховують затримки [8].

Окрім детермінованих алгоритмів, застосовуються також евристичні методи. Наприклад, використання генетичних алгоритмів дозволяє враховувати динамічні дані (GPS, реальний час) та оптимізувати маршрути в умовах невизначеності [2]. Такі підходи є особливо ефективними для адаптивного планування, обробки великих обсягів даних, задач із багатьма критеріями оптимізації. Сучасні транспортні системи потребують врахування кількох критеріїв одночасно: час прибуття, кількість пересадок, вартість, надійність. У цьому контексті широко застосовується концепція Парето-оптимальності, що дозволяє знаходити множину рішень без домінування одного критерію над іншими [5]. Додатково у роботах з оптимізації тарифів показано, що комбінування критеріїв (час + ціна) суттєво ускладнює задачу та потребує спеціалізованих моделей [7].

Найновіші дослідження демонструють застосування методів машинного навчання, зокрема глибинного підкріплювального навчання, для моделювання поведінки пасажирів та оптимізації маршрутів у мультимодальних мережах [4]. Такі підходи дозволяють враховувати поведінкові фактори, адаптувати маршрути до реальних умов, забезпечувати баланс між ефективністю та справедливістю розподілу ресурсів.

Викладення основного матеріалу і обґрунтування отриманих результатів.

Для ефективного управління життєвим циклом та контролем якості на етапі планування системи було обрано ітераційну модель розробки. Вона базується на гнучкій методології Scrum, що дозволило розбити процес проєктування на окремі логічні етапи (спринти). Такий підхід забезпечує гнучку реакцію на зміну вимог та своєчасне тестування ключових функцій інформаційної системи. Завдяки цьому етапи аналізу, проєктування та кодування виконувалися послідовно з можливістю постійного вдосконалення коду.

Архітектурне рішення розробленої програмної системи RailBooker ґрунтується на використанні шаблону Model-View-Template (MVT) [6], який забезпечує чітке розмежування між структурами даних, бізнес-логікою та користувацьким інтерфейсом. Взаємодія із системою здійснюється через веббраузер: клієнт надсилає запити на сервер, де вони обробляються за допомогою механізму маршрутизації URL. Для зберігання інформації про рейси та фінансові операції використовується централізована база даних, що спрощує керування даними. Захист доступу реалізовано із застосуванням сесій і токенів, що унеможливлює несанкціоноване використання персональної інформації.

Структуру системи організовано у вигляді окремих додатків (apps) у межах Django, кожен з яких відповідає за певну функціональну підсистему. Під час проєктування було розроблено структуру бази даних, яка включає ключові сутності: розклад руху, станції, рухомий склад, користувачів і замовлення. Ідентифікація користувачів та розмежування прав доступу реалізовані за допомогою стандартної підсистеми автентифікації Django (django.contrib.auth), що дозволило уникнути дублювання структур для зберігання облікових записів і ролей. Узгодженість даних підтримується через використання зовнішніх ключів, які забезпечують коректність операцій, зокрема при каскадному оновленні або видаленні записів. Така організація даних створює стабільну основу для виконання ключових процесів — реєстрації користувачів, пошуку маршрутів і керування бронюваннями через особистий кабінет.

Система передбачає шість основних режимів роботи, орієнтованих на різні категорії користувачів. Гостьовий режим призначений для неавторизованих відвідувачів і дозволяє переглядати розклад, здійснювати пошук рейсів, перевіряти наявність місць і виконувати бронювання. Режим пасажира розширює ці можливості, надаючи доступ до історії замовлень і персонального кабінету. Окремий режим передбачено для провідників: після службової авторизації вони можуть перевіряти квитки шляхом сканування QR-кодів. Режим касира забезпечує повноцінну роботу з квитками. Для керівника станції реалізовано функціонал керування розкладом і персоналом. Адміністративний режим надає розширені можливості управління станціями, потягами та працівниками системи. Програмна реалізація системи бронювання квитків RailBooker здійснена з використанням мови програмування Python та вебфреймворку Django, що забезпечує дотримання архітектурного патерну MVT.

Одним із найважливіших компонентів системи є алгоритм пошуку маршрутів, реалізований у функції find_routes_algoritm(). Метою роботи алгоритму є визначення всіх можливих маршрутів

між заданими початковою та кінцевою станціями на конкретну дату. Розглядаються два типи маршрутів: прямі (без пересадок) та маршрути з однією пересадкою. Вхідними даними є ідентифікатор початкової станції, ідентифікатор кінцевої станції, дата подорожі. Результатом роботи алгоритму є впорядкований список маршрутів із зазначенням часових характеристик та структури поїздки. Для забезпечення реалістичності маршрутів вводяться обмеження на час пересадки: мінімальний час пересадки — 5 хвилин, максимальний час пересадки — 12 годин. Ці обмеження визначають допустимий інтервал між прибуттям першого потяга та відправленням наступного.

На першому етапі алгоритм виконує пошук усіх рейсів, що проходять через початкову станцію у задану дату. Для кожного такого рейсу виконується наступний перелік кроків.

1. Отримання впорядкованого списку зупинок маршруту.
2. Визначення позицій початкової та кінцевої станцій у межах маршруту.
3. Перевірка коректності порядку проходження станцій (кінцева станція повинна слідувати після початкової).
4. Перевірка наявності часових характеристик (часу відправлення та прибуття).
5. Обчислення моментів відправлення та прибуття з урахуванням можливого переходу через північ (корекція дати прибуття).
6. Формування маршруту у разі успішного проходження всіх перевірок.

Таким чином, до результату додаються всі валідні прямі маршрути з відповідними часовими параметрами та тривалістю поїздки.

Другий етап алгоритму передбачає побудову маршрутів із використанням однієї пересадки та складається з кількох вкладених кроків.

Крок 1. Вибір першого сегмента маршруту (рис. 1). Формується множина всіх можливих початкових сегментів, тобто зупинок, на яких користувач може сісти на потяги у початковій станції. Для кожного сегмента перевіряється наявність часу відправлення, виключаються маршрути, які вже забезпечують пряме сполучення з кінцевою станцією, визначається відповідний рейс для заданої дати та обчислюється час відправлення.

```python
possible_first_segments = RouteStop.objects.filter(
    station=start_station,
    route__route_instances__date=search_date
).select_related('route', 'station')

for start_segment in possible_first_segments:
    # перевірка наявності часу відправлення
    if not start_segment.departing_time:
        continue

    route_1 = start_segment.route

    # виключення прямих маршрутів
    is_direct_route = RouteStop.objects.filter(
        route=route_1,
        station=end_station,
        stop_order__gt=start_segment.stop_order
    ).exists()
    if is_direct_route:
        continue

    # отримання рейсу на задану дату
    instance_1 = RouteInstance.objects.filter(
        route=route_1,
        date=search_date
    ).first()
    if not instance_1:
        continue

    # обчислення часу відправлення
    dep_dt_1 = datetime.combine(search_date, start_segment.departing_time)
```

Рис 1. Вибір першого сегмента маршруту

Крок 2. Визначення можливих точок пересадки (Рис. 2).

```
transfer_stops = RouteStop.objects.filter(
    route=route_1,
    stop_order__gt=start_segment.stop_order
).select_related('station')

for transfer_seg_1 in transfer_stops:
    transfer_station = transfer_seg_1.station

    # визначення часу прибуття
    if not transfer_seg_1.arrival_time:
        continue

    arr_dt_1 = get_valid_datetime(
        dep_dt_1.date(),
        transfer_seg_1.arrival_time,
        dep_dt_1
    )

    # допустимий інтервал пересадки
    min_dep_time_2 = arr_dt_1 + MIN_TRANSFER_TIME
    max_dep_time_2 = arr_dt_1 + MAX_TRANSFER_TIME
```

Рис 2. Визначення точок пересадки

Для кожного першого сегмента розглядаються всі наступні зупинки цього маршруту як потенційні станції пересадки. Для кожної такої станції визначається час прибуття та обчислюється допустимий інтервал для пересадки.

Крок 3. Пошук другого сегмента маршруту (Рис. 3).

```python
routes_from_transfer = RouteStop.objects.filter(
    station=transfer_station
).values_list('route', flat=True)

candidate_routes_2 = RouteStop.objects.filter(
    station=end_station,
    route__in=routes_from_transfer
).values_list('route', flat=True)

for route_id_2 in candidate_routes_2:
    # не той самий маршрут
    if route_id_2 == route_1.id:
        continue

    try:
        transfer_seg_2 = RouteStop.objects.get(
            route_id=route_id_2,
            station=transfer_station
        )
        end_seg_2 = RouteStop.objects.get(
            route_id=route_id_2,
            station=end_station
        )
    except RouteStop.DoesNotExist:
        continue

    # коректний порядок зупинок
    if transfer_seg_2.stop_order >= end_seg_2.stop_order:
        continue

    # перевірка часу
    if not transfer_seg_2.departing_time or not end_seg_2.arrival_time:
        continue

    # перевірка на надлишковість
    is_redundant = RouteStop.objects.filter(
        route_id=route_id_2,
        station=start_station,
        stop_order__gt=transfer_seg_2.stop_order
    ).exists()

    if is_redundant:
        continue
```

Рис. 3. Пошук другого сегмента маршруту

Для кожної станції пересадки виконується пошук другого маршруту, що задовольняє такі умови: проходить через станцію пересадки, забезпечує досягнення кінцевої станції, має коректний порядок зупинок (від станції пересадки до кінцевої), не є тим самим маршрутом, що і перший сегмент. Додатково виконується перевірка на надлишковість: якщо другий маршрут проходить через початкову станцію після точки пересадки, такий варіант відкидається як неефективний.

У разі виконання всіх умов:

- обчислюється час прибуття до кінцевої станції;
- визначається загальна тривалість подорожі;
- формується маршрут, що складається з двох сегментів (до пересадки та після неї);
- маршрут додається до списку результатів.

Особливістю алгоритму є коректна обробка часових значень у випадках, коли прибуття або відправлення відбувається після опівночі. Для цього використовується допоміжна функція, яка порівнює часи відправлення та прибуття, у разі необхідності коригує дату прибуття шляхом додавання однієї доби. Після завершення пошуку всі знайдені маршрути впорядковуються за двома критеріями: тип маршруту (прямі мають вищий пріоритет) та загальна тривалість поїздки (за зростанням). Це забезпечує відображення найзручніших маршрутів на початку списку.

Крок 4. Узгодження часових параметрів (Рис. 4).

```
for target_date in target_dates:
    instance_2 = RouteInstance.objects.filter(
        route_id=route_id_2,
        date=target_date
    ).first()

    if instance_2:
        # час відправлення другого поїзда
        potential_dep_dt_2 = datetime.combine(
            target_date,
            transfer_seg_2.departing_time
        )

        # перехід через північ
        if potential_dep_dt_2 < arr_dt_1:
            potential_dep_dt_2 += timedelta(days=1)

        # час очікування
        waiting_time = potential_dep_dt_2 - arr_dt_1

        # перевірка інтервалу
        if MIN_TRANSFER_TIME <= waiting_time <= MAX_TRANSFER_TIME:

            arr_dt_2 = get_valid_datetime(
                potential_dep_dt_2.date(),
                end_seg_2.arrival_time,
                potential_dep_dt_2
            )

            if arr_dt_2:
                total_duration = arr_dt_2 - dep_dt_1

                results.append({
                    'type': '3 пересадкою',
                    'min_price': 0,
                    'transfer_station': transfer_station.name,
                    'total_duration': total_duration,
                    'segments': [
                        {
                            'route': route_1,
                            'start_stop': start_segment,
                            'end_stop': transfer_seg_1,
                            'departure_datetime': dep_dt_1,
                            'arrival_datetime': arr_dt_1,
                            'trip_instance': instance_1
                        },
                        {
                            'route': instance_2.route,
                            'start_stop': transfer_seg_2,
                            'end_stop': end_seg_2,
                            'departure_datetime': potential_dep_dt_2,
                            'arrival_datetime': arr_dt_2,
                            'trip_instance': instance_2
                        }
                    ]
                })
                break
```

Рис. 4. Узгодження часових параметрів

Для кожного кандидата другого сегмента:

- перевіряється наявність рейсу у день прибуття або наступний день;
- обчислюється час відправлення з урахуванням можливого переходу через північ;
- визначається час очікування між потягами;
- перевіряється належність часу очікування до допустимого інтервалу.

На Рис. 5 продемонстровано вебінтерфейс із результатом роботи функції пошуку маршруту.

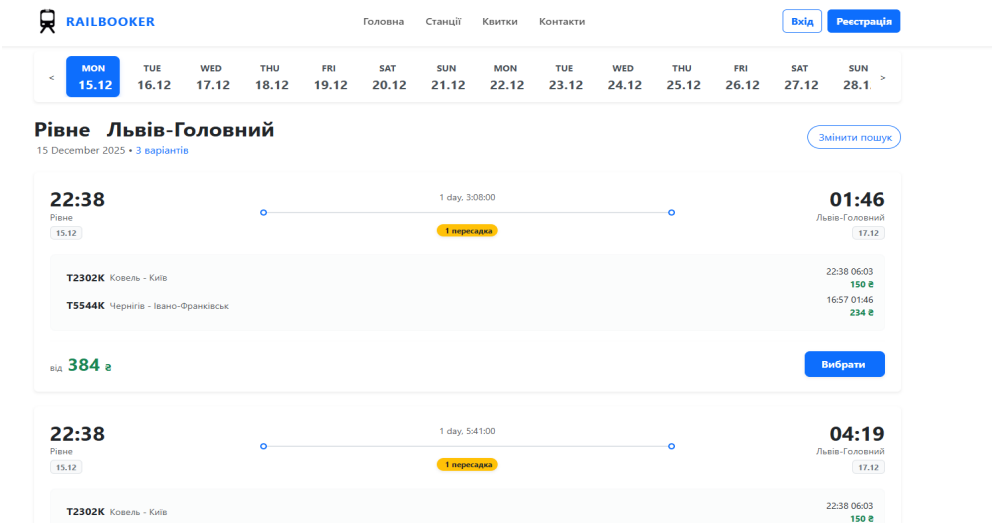

Рис. 5. Результат виводу функції find_routes_algoritm()

Тестування системи проводилося поетапно. Для цього було використано Django Test Framework. При тестуванні пошуку маршруту перевірялось, чи правильно система шукає маршрут з пересадкою та без із врахуванням мінімального часу на пересадку. Нижче наведено функції тестування find_routes_algoritm() (Рис. 6-8).

```
def test_find_direct_route(self):
    results = find_routes(self.station_a.id, self.station_c.id, self.search_date)

    direct_route = next((r for r in results if r['type'] == 'Прямий рейс'), None)
    self.assertIsNotNone(direct_route)
    self.assertEqual(direct_route['total_duration'], timedelta(hours=4))
    self.assertEqual(len(direct_route['segments']), 1)
```

Рис. 6. Перевірка пошуку маршруту без пересадки

```
def test_find_transfer_route(self):
    results = find_routes(self.station_a.id, self.station_c.id, self.search_date)

    transfer_route = next((r for r in results if r['type'] == "З пересадкою"), None)
    self.assertIsNotNone(transfer_route)
    self.assertEqual(transfer_route['transfer_station'], "Station B")
    self.assertEqual(len(transfer_route['segments']), 2)
```

Рис. 7. Перевірка пошуку маршрут з пересадкою

```
def test_min_transfer_time_fail(self):
    train_fast = Train.objects.create(name="999X", train_model="Hyperloop")
    route_fast = Route.objects.create(name="Fast Transfer", train=train_fast)

    RouteStop.objects.create(route=route_fast, station=self.station_b,
                             stop_order=1, departing_time=time(12, 4))
    RouteStop.objects.create(route=route_fast, station=self.station_c,
                             stop_order=2, arrival_time=time(13, 0))
    RouteInstance.objects.create(route=route_fast, date=self.search_date)

    results = find_routes(self.station_a.id, self.station_c.id, self.search_date)

    found = False
    for res in results:
        if res['type'] == "З пересадкою":
            if res['segments'][1]['route'].name == "Fast Transfer":
                found = True

    self.assertFalse(found, "Маршрут із пересадкою 4 хв не повинен бути знайдений")
```

Рис. 8. Перевірка пошуку маршруту з пересадкою менше ніж 5 хвилин

Висновки та перспективи подальших досліджень. Запропонований в роботі алгоритм дозволяє ефективно знаходити як прямі маршрути, так і маршрути з однією пересадкою з урахуванням часових обмежень, який враховує ключові особливості транспортних розкладів, зокрема необхідність контролю мінімального та максимального часу пересадки, а також коректну обробку ситуацій переходу через межі доби. Це забезпечує формування реалістичних і придатних до використання варіантів поїздок.

У межах роботи також розроблено веборієнтовану інформаційну систему, побудовану на основі архітектурного шаблону MVT, що забезпечує чітке розмежування між даними, бізнес-логікою та інтерфейсом користувача. Система реалізує повний цикл взаємодії з користувачем: від пошуку маршрутів до бронювання та оплати квитків. Проведене тестування підтвердило коректність роботи алгоритму як для прямих маршрутів, так і для маршрутів із пересадками, включаючи граничні випадки, наприклад, недостатній час для пересадки. Таким чином, розроблене програмне рішення забезпечує підвищення ефективності процесу планування поїздок, покращує якість обслуговування користувачів та може бути використане як основа для створення сучасних транспортних інформаційних систем.

Подальші дослідження можуть бути спрямовані на розширення функціональних можливостей алгоритму пошуку маршрутів та підвищення його ефективності. Одним із перспективних напрямів є реалізація підтримки маршрутів із кількома пересадками, що дозволить знаходити більш складні та гнучкі варіанти поїздок. Доцільним є також впровадження багатокритеріальної оптимізації, зокрема з урахуванням вартості квитків, тривалості подорожі, кількості пересадок та рівня надійності маршрутів. Це дозволить формувати множину Парето-оптимальних рішень і підвищити якість рекомендацій для користувача. Окрему увагу варто приділити інтеграції даних у реальному часі (затримки, зміни розкладу, завантаженість потягів), що дозволить адаптувати результати пошуку до поточних умов функціонування транспортної системи. Також перспективним є застосування методів машинного навчання для прогнозування попиту, моделювання поведінки пасажирів та персоналізації рекомендацій маршрутів. Крім того, подальший розвиток системи може включати оптимізацію продуктивності, масштабування для обробки великої кількості користувачів, а також розширення функціоналу інтерфейсу користувача з урахуванням сучасних вимог до вебзастосунків. Узагальнюючи, подальші дослідження можуть суттєво підвищити інтелектуальність, адаптивність та ефективність системи пошуку маршрутів.

Список бібліографічного опису

1. Доля К. В. Моделювання та прогнозування потоків пасажирів у мультимодальних маршрутних системах з урахуванням пересадок і надійності розкладів. *Central Ukrainian Scientific Bulletin Technical Sciences*. 2026. № 13(44). С. 382–389. URL: https://doi.org/10.32515/2664-262x.2026.13(44).382-389.
2. Basu, A., Raja, B., Gracious, R., & Vanajakshi, L. Dynamic trip planner for public transport using genetic algorithm. *Transport*. 2020. *35*(2), P. 156-167. https://doi.org/10.3846/transport.2020.12477.
3. Bożejko W., Grymin R., Pempera J. Scheduling and Routing Algorithms for Rail Freight Transportation. *Procedia Engineering*. 2017. Vol. 178. P. 206–212. URL: https://doi.org/10.1016/j.proeng.2017.01.098.
4. Chu, KF., Guo, W. Deep reinforcement learning of passenger behavior in multimodal journey planning with proportional fairness. *Neural Comput & Applic* 35, 20221–20240 (2023). https://doi.org/10.1007/s00521-023-08733-4
5. Delling D., Pajor T., Werneck R. F. Round-Based Public Transit Routing. *Transportation Science*. 2015. Vol. 49, no. 3. P. 591–604. URL: https://doi.org/10.1287/trsc.2014.0534.
6. Design and Deployment of Django-based Housing Information Management System / X. Yu et al. *Journal of Physics: Conference Series*. 2023. Vol. 2425, no. 1. P. 012018. URL: https://doi.org/10.1088/1742-6596/2425/1/012018.
7. Euler R., Lindner N., Borndörfer R. Price optimal routing in public transportation. *EURO Journal on Transportation and Logistics*. 2024. Vol. 13. P. 100128. URL: https://doi.org/10.1016/j.ejtl.2024.100128.
8. Katkalo D., Rohovyi A., Walsh T. Fast and Memory Efficient Multimodal Journey Planning with Delays. *https://arxiv.org/*. 2026. https://doi.org/10.48550/arXiv.2604.16149.
9. Path Planning of Rail-Mounted Logistics Robots Based on the Improved Dijkstra Algorithm / X. Zhou et al. *Applied Sciences*. 2023. Vol. 13, no. 17. P. 9955. URL: https://doi.org/10.3390/app13179955.
10. Rohovyi A., Abuaisha A., Walsh T. Early Pruning for Public Transport Routing. *https://arxiv.org/*. 2026. https://doi.org/10.48550/arXiv.2603.12592.

References

1. Dolia, K. V. Modeling and forecasting passenger flows in multimodal route systems considering transfers and schedule reliability. Central Ukrainian Scientific Bulletin. Technical Sciences, 2026, no. 13(44), pp. 382–389. Available at: https://doi.org/10.32515/2664-262x.2026.13(44).382-389.
2. Basu, A., Raja, B., Gracious, R., & Vanajakshi, L. Dynamic trip planner for public transport using genetic algorithm. *Transport*. 2020. *35*(2), P. 156-167. https://doi.org/10.3846/transport.2020.12477.
3. Bożejko W., Grymin R., Pempera J. Scheduling and Routing Algorithms for Rail Freight Transportation. *Procedia Engineering*. 2017. Vol. 178. P. 206–212. URL: https://doi.org/10.1016/j.proeng.2017.01.098.
4. Chu, KF., Guo, W. Deep reinforcement learning of passenger behavior in multimodal journey planning with proportional fairness. *Neural Comput & Applic* 35, 20221–20240 (2023). https://doi.org/10.1007/s00521-023-08733-4
5. Delling D., Pajor T., Werneck R. F. Round-Based Public Transit Routing. *Transportation Science*. 2015. Vol. 49, no. 3. P. 591–604. URL: https://doi.org/10.1287/trsc.2014.0534.
6. Design and Deployment of Django-based Housing Information Management System / X. Yu et al. *Journal of Physics: Conference Series*. 2023. Vol. 2425, no. 1. P. 012018. URL: https://doi.org/10.1088/1742-6596/2425/1/012018.
7. Euler R., Lindner N., Borndörfer R. Price optimal routing in public transportation. *EURO Journal on Transportation and Logistics*. 2024. Vol. 13. P. 100128. URL: https://doi.org/10.1016/j.ejtl.2024.100128.
8. Katkalo D., Rohovyi A., Walsh T. Fast and Memory Efficient Multimodal Journey Planning with Delays. *https://arxiv.org/*. 2026. https://doi.org/10.48550/arXiv.2604.16149.
9. Path Planning of Rail-Mounted Logistics Robots Based on the Improved Dijkstra Algorithm / X. Zhou et al. *Applied Sciences*. 2023. Vol. 13, no. 17. P. 9955. URL: https://doi.org/10.3390/app13179955.
10. Rohovyi A., Abuaisha A., Walsh T. Early Pruning for Public Transport Routing. *https://arxiv.org/*. 2026. https://doi.org/10.48550/arXiv.2603.12592.

Історія статті:
Отримано: 08.05.2026 Доопрацьовано: 19.05.2026 Прийнято до друку: 23.05.2026 Опубліковано: 29.05.2026