Додаток А
Код програми

```
# app/routers/distribution.py

from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from datetime import datetime
from app.database import get_db
from app import models

router = APIRouter()

@router.post("/auto-distribute")
def auto_distribute(db: Session = Depends(get_db)):
    unassigned_tasks = db.query(models.Task).filter(models.Task.assigned_user_id == None).all()
    if not unassigned_tasks:
        return {"message": "Немає непроасайнених завдань"}
    def get_user_load(user_id: int) -> int:
        return db.query(models.Task).filter(
            models.Task.assigned_user_id == user_id,
            models.Task.status != "Done"
        ).count()
    for task in unassigned_tasks:
        required_skills = db.query(models.TaskSkill).filter(models.TaskSkill.task_id == task.id).all()
        if not required_skills:
            all_users = db.query(models.User).all()
            if all_users:
                chosen_user = min(all_users, key=lambda u: get_user_load(u.id))
                task.assigned_user_id = chosen_user.id
            continue
        candidate_users = []
        all_users = db.query(models.User).all()
        for user in all_users:
            match_all_skills = True
            for req in required_skills:
                user_skill = db.query(models.UserSkill).filter(
                    models.UserSkill.user_id == user.id,
                    models.UserSkill.skill_id == req.skill_id
```

```
            ).first()
            if not user_skill:
                match_all_skills = False
                break
            if user_skill.proficiency_level < req.proficiency_level:
                match_all_skills = False
                break

        if match_all_skills:
            candidate_users.append(user)

    if not candidate_users:
        continue

    filtered_by_capacity = []
    for user in candidate_users:
        load = get_user_load(user.id)
        if load < user.capacity:
            filtered_by_capacity.append(user)

    if not filtered_by_capacity:
        continue

    chosen_user = min(filtered_by_capacity, key=lambda u: get_user_load(u.id))

    task.assigned_user_id = chosen_user.id

db.commit()

return {"message": "Автоматизований розподіл завдань виконано успішно!"}
```

Код програми для взаємодії користувача з базою даних

```
# app/crud.py

from sqlalchemy.orm import Session
from datetime import datetime
from app import models, schemas

def get_all_users(db: Session):
    return db.query(models.User).all()

def get_user_by_id(db: Session, user_id: int):
    return db.query(models.User).filter(models.User.id == user_id).first()

def create_user(db: Session, user_data: schemas.UserCreate):
    db_user = models.User(
        name=user_data.name,
        email=user_data.email,
        is_active=user_data.is_active,
        created_at=datetime.utcnow(),
        updated_at=None,
        role=user_data.role,
        capacity=user_data.capacity,
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    if user_data.skills:
        for skill_link in user_data.skills:
            us = models.UserSkill(
                user_id=db_user.id,
                skill_id=skill_link.skill_id,
                proficiency_level=skill_link.proficiency_level
            )
            db.add(us)
        db.commit()
        db.refresh(db_user)
```

```
    return db_user

def update_user_full(db: Session, user_id: int, user_data: schemas.UserFullUpdate):
    db_user = db.query(models.User).filter(models.User.id == user_id).first()
    if not db_user:
        return None

    db_user.name = user_data.name
    db_user.email = user_data.email
    db_user.is_active = user_data.is_active
    db_user.updated_at = datetime.utcnow()
    db_user.role = user_data.role
    db_user.capacity = user_data.capacity

    db.commit()
    db.refresh(db_user)

    db.query(models.UserSkill).filter(models.UserSkill.user_id == db_user.id).delete()
    db.commit()

    for skill_link in user_data.skills:
        us = models.UserSkill(
            user_id=db_user.id,
            skill_id=skill_link.skill_id,
            proficiency_level=skill_link.proficiency_level
        )
        db.add(us)

    db.commit()
    db.refresh(db_user)
    return db_user

def patch_user(db: Session, user_id: int, user_data: schemas.UserUpdate):
    db_user = db.query(models.User).filter(models.User.id == user_id).first()
    if not db_user:
        return None

    to_update = user_data.dict(exclude_unset=True)
```

```
    if "name" in to_update:
        db_user.name = to_update["name"]
    if "email" in to_update:
        db_user.email = to_update["email"]
    if "is_active" in to_update:
        db_user.is_active = to_update["is_active"]
    if "role" in to_update:
        db_user.role = to_update["role"]
    if "capacity" in to_update:
        db_user.capacity = to_update["capacity"]

    db_user.updated_at = datetime.utcnow()
    db.commit()
    db.refresh(db_user)

    if "skills" in to_update and to_update["skills"] is not None:
        db.query(models.UserSkill).filter(models.UserSkill.user_id ==
db_user.id).delete()
        db.commit()

        for skill_link in to_update["skills"]:
            us = models.UserSkill(
                user_id=db_user.id,
                skill_id=skill_link.skill_id,
                proficiency_level=skill_link.proficiency_level
            )
            db.add(us)

        db.commit()
        db.refresh(db_user)

    return db_user

def delete_user(db: Session, user_id: int):
    db_user = db.query(models.User).filter(models.User.id == user_id).first()
    if not db_user:
        return None
    db.delete(db_user)
```

```
    db.commit()
    return db_user

def get_all_skills(db: Session):
    return db.query(models.Skill).all()

def get_skill_by_id(db: Session, skill_id: int):
    return db.query(models.Skill).filter(models.Skill.id == skill_id).first()

def create_skill(db: Session, skill_data: schemas.SkillCreate):
    db_skill = models.Skill(name=skill_data.name)
    db.add(db_skill)
    db.commit()
    db.refresh(db_skill)
    return db_skill

def update_skill_full(db: Session, skill_id: int, skill_data: schemas.SkillFullUpdate):
    db_skill = db.query(models.Skill).filter(models.Skill.id == skill_id).first()
    if not db_skill:
        return None

    db_skill.name = skill_data.name
    db.commit()
    db.refresh(db_skill)
    return db_skill

def patch_skill(db: Session, skill_id: int, skill_data: schemas.SkillPatchUpdate):
    db_skill = db.query(models.Skill).filter(models.Skill.id == skill_id).first()
    if not db_skill:
        return None

    to_update = skill_data.dict(exclude_unset=True)
    if "name" in to_update:
        db_skill.name = to_update["name"]

    db.commit()
    db.refresh(db_skill)
    return db_skill
```

```
def delete_skill(db: Session, skill_id: int):
    db_skill = db.query(models.Skill).filter(models.Skill.id == skill_id).first()
    if not db_skill:
        return None
    db.delete(db_skill)
    db.commit()
    return db_skill

def get_all_tasks(db: Session):
    return db.query(models.Task).all()

def get_task_by_id(db: Session, task_id: int):
    return db.query(models.Task).filter(models.Task.id == task_id).first()

def create_task(db: Session, task_data: schemas.TaskCreate):
    db_task = models.Task(
        title=task_data.title,
        description=task_data.description,
        priority=task_data.priority,
        status=task_data.status,
        deadline=task_data.deadline,
        created_at=datetime.utcnow(),
        updated_at=None,
        comlexity=task_data.comlexity,
        estimated_hours=task_data.estimated_hours
    )
    db.add(db_task)
    db.commit()
    db.refresh(db_task)

    # Додаємо skills
    if task_data.skills:
        for skill_link in task_data.skills:
            ts = models.TaskSkill(
                task_id=db_task.id,
                skill_id=skill_link.skill_id,
                proficiency_level=skill_link.proficiency_level
            )
            db.add(ts)
```

```
        db.commit()
        db.refresh(db_task)

    return db_task

def update_task_full(db: Session, task_id: int, task_data: schemas.TaskFullUpdate):
    db_task = db.query(models.Task).filter(models.Task.id == task_id).first()
    if not db_task:
        return None

    db_task.title = task_data.title
    db_task.description = task_data.description
    db_task.priority = task_data.priority
    db_task.status = task_data.status
    db_task.deadline = task_data.deadline
    db_task.assigned_user_id = task_data.assigned_user_id
    db_task.updated_at = datetime.utcnow()
    db_task.comlexity = task_data.comlexity
    db_task.estimated_hours = task_data.estimated_hours

    db.commit()
    db.refresh(db_task)

    db.query(models.TaskSkill).filter(models.TaskSkill.task_id == db_task.id).delete()
    db.commit()

    for skill_link in task_data.skills:
        ts = models.TaskSkill(
            task_id=db_task.id,
            skill_id=skill_link.skill_id,
            proficiency_level=skill_link.proficiency_level
        )
        db.add(ts)
    db.commit()
    db.refresh(db_task)

    return db_task

def patch_task(db: Session, task_id: int, task_data: schemas.TaskPatchUpdate):
```

```
    db_task = db.query(models.Task).filter(models.Task.id == task_id).first()
    if not db_task:
        return None

    to_update = task_data.dict(exclude_unset=True)
    if "title" in to_update:
        db_task.title = to_update["title"]
    if "description" in to_update:
        db_task.description = to_update["description"]
    if "priority" in to_update:
        db_task.priority = to_update["priority"]
    if "status" in to_update:
        db_task.status = to_update["status"]
    if "deadline" in to_update:
        db_task.deadline = to_update["deadline"]
    if "assigned_user_id" in to_update:
        db_task.assigned_user_id = to_update["assigned_user_id"]
    if "comlexity" in to_update:
        db_task.comlexity = to_update["comlexity"]
    if "estimated_hours" in to_update:
        db_task.estimated_hours = to_update["estimated_hours"]

    db_task.updated_at = datetime.utcnow()
    db.commit()
    db.refresh(db_task)

    if "skills" in to_update and to_update["skills"] is not None:
        db.query(models.TaskSkill).filter(models.TaskSkill.task_id ==
db_task.id).delete()
        db.commit()
        for skill_link in to_update["skills"]:
            ts = models.TaskSkill(
                task_id=db_task.id,
                skill_id=skill_link.skill_id,
                proficiency_level=skill_link.proficiency_level
            )
            db.add(ts)
        db.commit()
        db.refresh(db_task)
```

```
    return db_task

def delete_task(db: Session, task_id: int):
    db_task = db.query(models.Task).filter(models.Task.id == task_id).first()
    if not db_task:
        return None
    db.delete(db_task)
    db.commit()
    return db_task
```

Додаток Б Апробація результатів

АНАЛІЗ СУЧАСНИХ СИСТЕМ АВТОМАТИЗОВАНОГО РОЗПОДІЛУ ЗАВДАНЬ У ВИРОБНИЦТВІ

Бацуля Р.В.
Харківський національний університет радіоелектроніки
Україна, Харків, пр. Науки. 14

Дослідження присвячено аналізу сучасних підходів до автоматизованого розподілу завдань у виробничих системах, що є критично важливими для підвищення продуктивності, зменшення простоїв та оптимального використання ресурсів. У зв'язку зі зростаючою складністю виробничих процесів та впровадженням концепції Industry 4.0, інтеграція автоматизації набуває стратегічного значення. У статті розглянуто рівні автоматизації, методи оптимізації розподілу ресурсів, а також перспективи використання людино-роботизованих систем для підтримки адаптивного і гнучкого управління завданнями. Особливу увагу приділено застосуванню евристичних алгоритмів та генетичних методів, які сприяють ефективному розподілу завдань у динамічному середовищі, забезпечуючи пристосування до змінних умов виробництва та мінімізуючи втрати часу. Сучасні технології дозволяють досягти високої ефективності та якості виробничих процесів, відповідаючи на новітні вимоги до гнучкості та надійності систем в умовах індустрії майбутнього.

Ключові слова: task allocation, resource optimisation, human-robot system, industry 4.0.

ANALYSIS OF MODERN SYSTEMS FOR AUTOMATED TASK ALLOCATION IN MANUFACTURING

Batsulia R.
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av., 14

The study focuses on the analysis of modern approaches to automated task allocation in production systems, which is crucial for increasing productivity, reducing downtime and optimising resource use. With the increasing complexity of manufacturing processes and the implementation of the Industry 4.0 concept, the integration of automation has gained strategic importance. This article examines the levels of automation, methods for optimising resource allocation, and the prospects for using human-robot collaborative systems to support adaptive and flexible task management. Special attention is given to heuristic algorithms and genetic methods that facilitate effective task allocation in dynamic environments, ensuring adaptability to changing production conditions and minimising time losses. Modern technologies enable high efficiency and quality in manufacturing processes, meeting the latest demands for flexibility and reliability in future-oriented industries.

Keywords: task allocation, resource optimisation, human-robot system, industry 4.0.

АКТУАЛЬНІСТЬ РОБОТИ. У сучасному виробничому середовищі зростає потреба в ефективних системах автоматизованого розподілу завдань, які здатні адаптуватися до швидких змін умов виробництва, нестабільності ресурсів та зростаючої складності процесів. Впровадження концепції Industry 4.0 сприяє інтеграції новітніх технологій, таких як роботизовані системи, штучний інтелект та алгоритми машинного навчання, що робить автоматизацію розподілу завдань стратегічно важливою для забезпечення конкурентоспроможності підприємств. Автоматизація управління завданнями у виробничих системах не лише підвищує продуктивність, але й забезпечує зниження витрат та втрат через простої, сприяє оптимальному використанню ресурсів та дозволяє реалізувати гнучке управління. Використання методів штучного інтелекту, зокрема генетичних алгоритмів та евристичних методів, відкриває нові можливості для створення динамічних систем, які можуть ефективно пристосовуватись до змінних умов і вимог. У цьому контексті дослідження сучасних підходів до автоматизованого розподілу завдань у виробничих групах набуває особливої актуальності, адже воно сприяє подальшому розвитку адаптивних та самонавчальних систем управління, що відповідають вимогам індустрії майбутнього.

Метою дослідження є аналіз та розробка підходів до автоматизованого розподілу завдань у виробничих системах з метою підвищення ефективності, гнучкості та продуктивності виробництва. Дослідження спрямоване на вивчення сучасних методів оптимізації розподілу ресурсів, оцінку рівнів автоматизації та застосування інтелектуальних алгоритмів, таких як генетичні та евристичні алгоритми, для досягнення динамічного управління завданнями. Особлива увага приділяється інтеграції людино-роботизованих систем, які дозволяють адаптувати виробничі процеси до змінних умов, знижуючи витрати часу на простої та забезпечуючи ефективне використання наявних ресурсів.

У багатокритеріальному розкладі автоматизованих виробничих систем (AMS) розв’язання задач оптимізації вимагає одночасного врахування ефективного використання ресурсів та уникнення тупиків. Одним із підходів для досягнення цієї мети є використання генетичних алгоритмів на основі Парето-оптимізації, що дозволяють знаходити збалансовані рішення для складних виробничих процесів. В моделюванні системи використано мережі Петрі, які ефективно описують динаміку розподілу ресурсів, обмеження їх доступності та можливі конфлікти. Такий підхід дозволяє уникати тупикових станів, забезпечуючи безперервність виробничого процесу. Генетичний алгоритм, запропонований для розкладу завдань, складається з перевірки життєздатності рішень та виправлення неефективних розподілів ресурсів, що допомагає уникнути тупиків на етапі планування. Для кожного індивіда (потенційного рішення) проводиться перевірка на відповідність обмеженням AMS і корекція шляхом контролю доступності ресурсів. У моделі розглядаються як статичні, так і динамічні варіанти розкладу, а завдяки адаптації за методом Парето забезпечується висока якість оптимізації у багатокритеріальному середовищі (рис. 1).

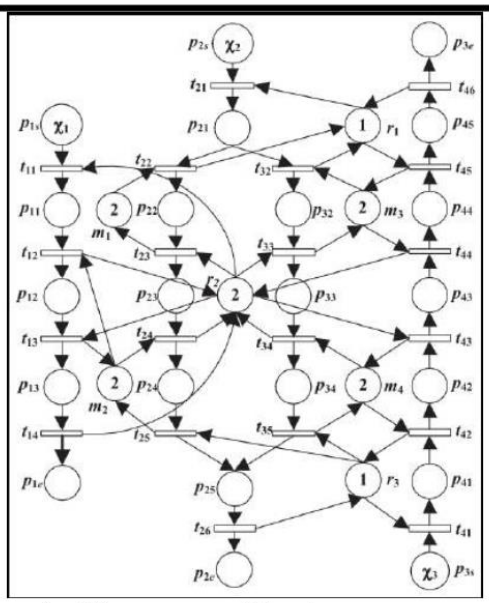

Рисунок 1 – Модель мережі Петрі для автоматизованої виробничої системи (AMS)

Висновки підтверджують, що алгоритм є ефективним у вирішенні задач багатокритеріального розкладу та дозволяє знизити витрати часу й ресурсів завдяки урахуванню різних показників ефективності [1].

Сучасні стратегії динамічного розподілу завдань у мультиагентних системах роботів є критично важливими для підвищення продуктивності та адаптивності в різних середовищах. На відміну від статичних методів, що підходять для передбачуваних і повторюваних процесів, динамічні методи здатні ефективно адаптуватися до змінних умов. В огляді розглядаються ключові підходи, серед яких методи, засновані на ринкових аукціонах, оптимізаційні стратегії, поведінковий підхід та кластеризація завдань. Методи ринкових аукціонів імітують принцип торгів, де роботи змагаються за виконання завдань, що підвищує ефективність у системах з централізованим управлінням. Оптимізаційні стратегії часто використовують алгоритми, як-от генетичні та частинкового рою, які вирішують задачі мінімізації часу або витрат ресурсів, зокрема для складних і динамічних умов. Поведінковий підхід робить розподіл завдань більш гнучким, адаптуючи поведінку роботів до різних обмежень та непередбачених ситуацій. Кластеризація завдань, навпаки, групує подібні або близькі завдання, що зменшує загальну відстань пересування роботів і підходить для задач типу пошуку та порятунку. В огляді визначено, що хоча існує багато ефективних підходів, питання інтеграції цих стратегій та адаптації до змінних умов залишається актуальним для подальших досліджень [2].

Методологія розподілу завдань у спільних командах людей і роботів ґрунтується на оцінці складності завдань в умовах збірки. Враховуються фізичні характеристики компонентів і умови виконання завдань, що дозволяє визначити

автоматизаційний потенціал кожного завдання. Оцінка складності включає такі аспекти, як зручність захоплення компонентів, механізм подачі, методи монтажу та вимоги до безпеки (рис. 2).

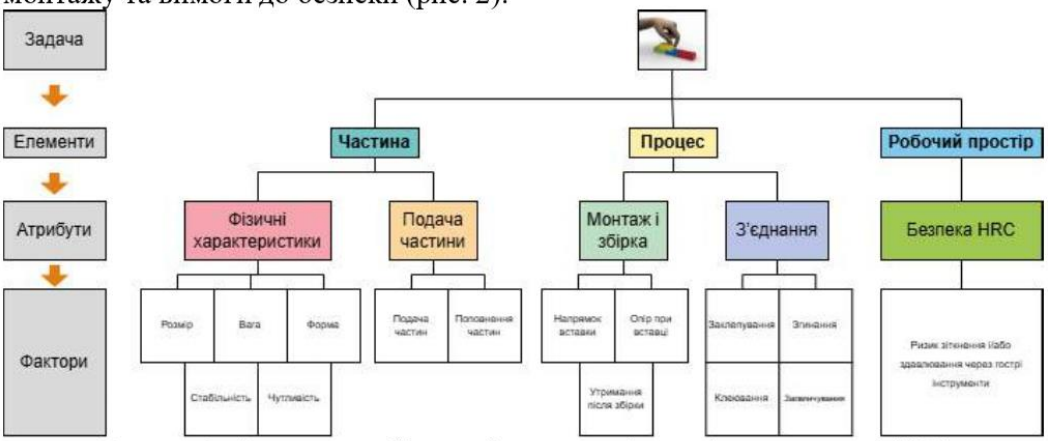

Рисунок 2 – Завдання на збірку та його складові для оцінки складності HRC

Цей підхід дозволяє відокремити завдання, що потребують участі людини, від тих, які можуть бути виконані роботами, таким чином забезпечуючи оптимальний баланс навантаження в команді. Такий підхід до класифікації завдань значно зменшує час налаштування системи та сприяє легшій інтеграції роботів у виробниче середовище [3].

Динамічне призначення завдань у команді людина-робот для спільних виробничих осередків базується на класифікації завдань і розподілі робіт відповідно до специфічних навичок обох учасників. Метод передбачає три основні етапи: визначення ключових індикаторів для кожного завдання (вага, переміщення, точність, спритність), класифікацію завдань за придатністю для виконання людиною, роботом або обома, а також остаточне призначення завдань із урахуванням тривалості та послідовності виконання. Використання цього підходу дозволяє гнучко керувати розподілом завдань, підтримуючи динамічне перерозподілення завдань у разі затримок або змін у процесі. Це забезпечує адаптивність у середовищах із змінними умовами, зменшуючи ризики простоїв і оптимізуючи навантаження між людиною і роботом [4].

Динамічна стратегія розподілу завдань у системах спільної збірки в контексті Industry 5.0 дозволяє поєднати продуктивність із благополуччям оператора. Врахування людського фактора, як-от ергономіка, ментальне навантаження, навички й можливості, є пріоритетом для оптимізації робочого середовища, оскільки впливає на продуктивність та задоволення оператора. Ця стратегія передбачає перерозподіл завдань між оператором і роботом у реальному часі, щоб підтримувати баланс робочого навантаження залежно від стану оператора, наприклад, коли його енергетичне навантаження стає надто високим. Підхід оптимізує як час виконання завдань, так і витрати енергії оператора, завдяки чому система адаптується до змін у потребах людини. Така стратегія сприяє зниженню стресу оператора й забезпечує гнучкість у розподілі завдань, що відповідає вимогам Industry 5.0 до створення гармонійного

робочого середовища, де продуктивність поєднується із комфортом та безпекою [5].

Система розподілу завдань у виробничих системах передбачає вимірювання та аналіз рівнів автоматизації, що забезпечує баланс між фізичними та когнітивними завданнями для операторів. У моделі використовуються різні рівні автоматизації (LoA), які допомагають визначити ступінь автоматизації як для фізичних, так і для когнітивних завдань, надаючи можливість адаптувати їх для конкретних вимог виробництва. Модель також враховує рівень інформаційного забезпечення (LoI) та рівень компетентності операторів (LoC), що дає змогу ефективно розподіляти завдання між людиною та машиною залежно від ситуації. Такий підхід сприяє оптимізації робочого процесу, дозволяючи операторам зосередитися на завданнях, що потребують їхньої уваги, тоді як менш складні або повторювані завдання виконують машини [6].

Метод розподілу завдань у виробничих системах "людина-машина" на основі глибокого навчання з підкріпленням забезпечує динамічний розподіл завдань з урахуванням факторів, як-от компетентність оператора та накопичення втоми (рис. 3). Підхід дозволяє обирати найбільш відповідного виконавця для кожного завдання залежно від його продуктивності та наявних обмежень, таких як черга завдань, доступність обладнання та ризик перевантаження оператора. У процесі навчання модель враховує накопичені дані, що дозволяє агенту враховувати приховані характеристики операторів, такі як втома, яка може впливати на швидкість та якість виконання завдань [7].

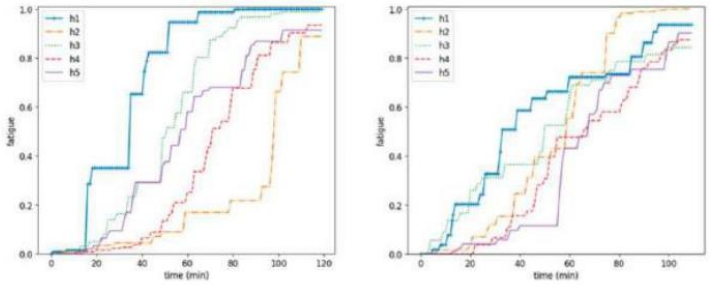

Рисунок 3 – Накопичення втоми: до тренування (ліворуч) і після тренування (праворуч)

Взаємодія людини і робота у виробничих системах потребує ефективного розподілу завдань для оптимізації робочого процесу та підвищення продуктивності. Розглядаються два основні підходи: статичний і динамічний розподіл завдань. Статичний підхід передбачає попередній розподіл завдань, що підходить для передбачуваних і повторюваних процесів. Натомість динамічний підхід дозволяє реорганізацію завдань у реальному часі, враховуючи зміни в умовах, навантаженні та стані як людини, так і робота. При динамічному розподілі завдань ключовими факторами є когнітивне навантаження, фізичні обмеження та особливості завдань, що дозволяє ефективно адаптувати процес до вимог Industry 4.0. Окрім продуктивності, важливими критеріями є також

безпека та зручність працівників, що сприяє гармонізації співпраці людини і робота [8].

Розподіл завдань у мульти-роботній системі з використанням ресурсного обміну та динамічного порогового підходу сприяє підвищенню адаптивності системи до змінних умов середовища, таких як виникнення несправностей або дефіцит ресурсів. У системі впроваджено порогове значення, визначене центральною одиницею, яке слугує орієнтиром для прийняття рішень щодо виконання завдань, навіть у випадках нестачі ресурсів. Якщо рівень ресурсів робота не дозволяє виконати завдання самостійно, він ініціює аукціон для отримання бракуючих ресурсів від інших роботів у системі, що забезпечує гнучкість у використанні доступних засобів. Динамічне оновлення порогового значення на основі загального статусу системи дозволяє роботам швидко адаптуватися до поточних умов, мінімізуючи простої та оптимізуючи завантаження. У результаті, система досягає високого рівня ефективності як у плані часу виконання завдань, так і в плані використання ресурсів. Такий підхід забезпечує досягнення високого рівня продуктивності та дозволяє використовувати ресурси на 96%, що підтверджено теоретично та експериментально [9].

Методика динамічного розподілу завдань у гетерогенній розподіленій системі використовує мульти-евристичний генетичний алгоритм, який поєднує вісім загальних евристик для оптимального розподілу завдань між процесорами. Цей підхід спрямований на мінімізацію загального часу виконання завдань і здатен адаптуватися до постійно змінюваних ресурсів у системі. Генетичний алгоритм створює початкову популяцію, використовуючи кілька евристик, що покращує початкові рішення в порівнянні зі звичайною випадковою ініціалізацією. Адаптивний підхід, який враховує доступність процесорів і ресурсів у реальному часі, забезпечує ефективність навіть при високій гетерогенності середовища. Результати експериментів показують, що запропонований алгоритм перевищує за ефективністю інші сучасні алгоритми планування для розподілених систем, особливо у випадках високої гетерогенності ресурсів [10].

Метод розподілу незалежних завдань у мультипроцесорних системах за допомогою евристичного генетичного алгоритму забезпечує оптимізацію часу виконання завдань шляхом поєднання двох класичних алгоритмів зі списковим розподілом. У цьому підході використовуються "Longest Processing Time" (LPT) та "Shortest Processing Time" (SPT) для формування початкової популяції генетичного алгоритму, що дозволяє краще розподілити завдання між процесорами і скоротити час виконання. Генетичний алгоритм забезпечує пріоритетність завдань на основі часу їх виконання і використовує процеси відбору, схрещування та мутації для досягнення оптимального графіка. За результатами експериментів, запропонований алгоритм перевершує традиційні методи планування, демонструючи кращі показники ефективності для мультипроцесорних середовищ [11].

ВИСНОВКИ: Різноманітність сучасних підходів до автоматизованого розподілу завдань у виробничих системах демонструє важливість оптимізації

робочих процесів у контексті Industry 4.0. Використання генетичних алгоритмів, мереж Петрі та підходів на основі штучного інтелекту забезпечує адаптивність і ефективність у розподілі завдань. Виявлено, що динамічні методи розподілу, такі як алгоритми з аукціонами ресурсів та евристичні методи, забезпечують вищу гнучкість і швидкість адаптації систем до змінних умов виробництва, зменшуючи час простоїв та підвищуючи ефективність використання ресурсів. Загалом, впровадження таких методів у виробничих процесах сприяє створенню гнучких і самонавчальних систем, які відповідають сучасним вимогам до продуктивності та надійності.

Список використаних джерел

1. A Pareto-based genetic algorithm for multi-objective scheduling of automated manufacturing systems [Електронний ресурс]. – режим доступу https://www.researchgate.net/publication/338408247_A_Pareto-based_genetic_algorithm_for_multi-objective_scheduling_of_automated_manufacturing_systems.
2. Review on state-of-the-art dynamic task allocation strategies for multiple-robot systems [Електронний ресурс]. – режим доступу https://www.researchgate.net/publication/344180532_Review_on_state-of-the-art_dynamic_task_allocation_strategies_for_multiple-robot_systems.
3. Complexity-based task allocation in human-robot collaborative assembly [Електронний ресурс]. – режим доступу https://d1wqtxts1xzle7.cloudfront.net/95120376/pdf-libre.pdf?1669904006=&response-content-disposition=inline%3B+filename%3DComplexity_based_task_allocation_in_huma.pdf&Expires=1731185002&Signature=AEawOJxnrySouq5vkAYD9-nCeS4mvAuvR2gtkWzD4di8QVDkxbG654iSvOqwUxMvsm0blX~10SOGydYEa8OouuFLoCVekYEFKrU6gCbMrBmWxKOrJkhOOMp9egdBZGexCpeWfbUjPjOg265seVk8OBrZ82MpKNNkrt7uIPslmy2D00MIczE4ZwwzQ-pkC-g7I0-e~vbPgvMKG8J4oL~WRLJj~lh6Ano~SD7Gi0zy-6WXMccMr3AkzJlPMTSK5uT3KnzPMzRCBK8yZ1DJmLR1CofVzFlI8mGlRxErFvWK9MqdzQ1ZALYB0JevsyPKfLRzjHdxU60Ioa1L3R7KPfJ9Lg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.
4. Dynamic task classification and assignment for the management of human-robot collaborative teams in workcells [Електронний ресурс]. – режим доступу http://www.qualityengineering.polito.it/content/download/394/2233/file/Dynamic%20task%20classification%20and%20assignment%20for%20the%20management.pdf.
5. Achieving productivity and operator well-being a dynamic task allocation strategy for collaborative assembly systems in Industry 5.0 [Електронний ресурс]. – режим доступу https://www.researchgate.net/publication/383528874_Achieving_productivity_and_operator_well-

being_a_dynamic_task_allocation_strategy_for_collaborative_assembly_systems_in_Industry_50.
6. Task Allocation in Production Systems - Measuring and Analysing Levels of Automation [Електронний ресурс]. – режим доступу https://www.sciencedirect.com/science/article/pii/S1474667016331007.
7. Task Allocation in Human–Machine Manufacturing Systems Using Deep Reinforcement Learning [Електронний ресурс]. – режим доступу https://www.mdpi.com/2071-1050/14/4/2245
8. Review of task allocation for human-robot collaboration in assembly [Електронний ресурс]. – режим доступу https://www.tandfonline.com/doi/full/10.1080/0951192X.2023.2204467#d1e155.
9. Task allocation in multi-robot system using resource sharing with dynamic threshold approach [Електронний ресурс]. – режим доступу https://pmc.ncbi.nlm.nih.gov/articles/PMC9067701/.
10. Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system [Електронний ресурс]. – режим доступу https://www.sciencedirect.com/science/article/pii/S0743731510000705.
11. A Heuristic Genetic Algorithm for Independent Task Scheduling [Електронний ресурс]. – режим доступу https://www.researchgate.net/publication/331629356_A_Heuristic_Genetic_Algorithm_for_Independent_Task_Scheduling.

АНАЛІЗ МОБІЛЬНИХ РОБОТІВ ДЛЯ РОЗПІЗНАВАННЯ СКЛАДНИХ КОНФІГУРАЦІЙ ДЕТАЛЕЙ ПРИЛАДОБУДІВНОГО ВИРОБНИЦТВА

Рудакова Г.В.
Харківський національний університет радіоелектроніки
Україна, Харків, пр. Науки. 14

Дослідження розглядає актуальні методи та підходи до розробки мобільних роботів, здатних розпізнавати складні конфігурації деталей у приладобудівному виробництві. Сучасні виробничі процеси вимагають високого рівня автоматизації для забезпечення точної і швидкої обробки складних деталей, що включає використання роботів з інтелектуальними системами комп'ютерного зору та машинного навчання. Із зростанням технологічних вимог у виробництві мобільні роботи набувають важливого значення завдяки здатності виконувати навігацію в динамічному середовищі та розпізнавати деталі за допомогою нейронних мереж і алгоритмів глибокого навчання, що підвищує ефективність виробничих операцій. Використання комп'ютерного зору, зокрема згорткових нейронних мереж (CNN) та технологій

Додаток В

Демонстраційний матеріал

Формат	Позиція	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	Примітки
			Текстові документи		
А4	1	ГЮІК. 441240.020 ПЗ	Пояснювальна записка	1	93 с.
			Інші документи		
	2		Пояснювальна записка	1	Ел. варіант
			Презентаційний матеріал		10 с.

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК.441240.020 ПЗ		
Розроб.		Бацуля Р.В.			Розробка системи автоматизованого розподілу завдань виробничих групах	Літера	Аркуш	Аркуші
Перевір.		Бронніков А.І.				У	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Демська Н.П.						
Затв.		Невлюдов І.Ш.						