Exergic balance diagram of system operation with hydraulic distributor is given in Picture 4.

Consequently, the variant with hydraulic distributor ensures a better use of exergy which results in higher thermodynamic indexes.

When the cold is transferred to deeper horizons with high pressure thermal exchanger, the cold is transferred by means of recuperation, whereas with hydraulic distributor it goes through the latter directly to air coolers in the bottomholes. By means of this, effective air cooling in the bottomhole is achieved and energy resources can be saved.

Heat engineering comparison of the variants of mine air conditioning system operation with high pressure thermal exchanger and a hydraulic distributor has shown that while working with high pressure thermal exchanger η = 18,49 %, whereas the same for hydraulic distributor is η = 98,55 %.

Conclusion.1. In the process of cold transition to deep horizons with the help of high pressure thermal exchanger, the cold is conveyed in a recuperative way, whereas with hydraulic distributor it goes through the distributor it is conveyed directly to air-cooling unit in the bottomhole. By means of this effective air cooling is achieved.

2. A very small exergy loss is registered while cold bearer transition to deep horizons by hydraulic distributor. In such way energy resources are used more effectively.

3. Thus, it can be observed that the operation of the system with hydraulic distributor will be more efficient, with smaller exergy losses for cold conveyed to air cooling units.

4. Thermal technical correlation of performed for different variants of mine air conditioning with high-pressure thermal exchanger and hydraulic distributor has shown that while working with η = 18,49 % and with hydraulic distributor it is η = 98,55 %.

REFERENCES

1. Averin F. A. Laboratornyj praktikum rudnichnoj ventiljacii / F. A. Averin, V. A. Bojko, V. A. Dolinskij. – M. : Nedra, 1966. – 64 s .

2. A. s. 1642204, MKI F24 F13/06. Vozduhoraspredelitel' / V. F. Rozhko, V. A. Steblovcev, I. S. Ignashkin (SSSR). – № 4678905/29; Zajavl. 14.04.89; Opubl. 15.04.91, Bjul. №14. – 3 s.

3. Andrjushhenko A. I. Termodinamicheskie raschjoty optimal'nyh parametrov teplovyh jelektrostancij. – M. : Vyssh. shk., 1963. – 230 s.

4. Andrjushhenko A. I. Osnovy termodinamicheskih ciklov teplojenergeticheskih ustanovok. – M. : Vyssh. shk., 1968. – 288 s.

5. Andrjushhenko V. N. Metody izmerenija sostojanija mikroklimata v gornyh vyrabotkah / V. N. Andrjushhenko, V. F. Rozhko. – M. : 1989. – 16 s. – Dek v CNIIJeI – ugol' 28.11.1989, №5011.

6. Brodjanskij V. M. Jeksergeticheskij metod termodinamicheskogo analiza. – M. : Jenergija 1973. – 296 s.7. Pat. 23867 A Ukraina, UA E21 F3/00 Gidrorozpodilnik / V. F. Rozhko, I. S. Ignashkin, L. O. Haruk, F. O. Korsun ta in. / Opubl. 31.08.98. – Byul. № 4. – S. 73

УДК 004.896:621.796.5

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ПОШУКОВИХ АЛГОРИТМІВ У СИСТЕМІ УПРАВЛІННЯ СКЛАДСЬКИМ ПРИМІЩЕННЯМ

В. О. Ужеловський, к. т. н., доц., П. С. Кашуба, магістр

Ключові слова: *алгоритм, імітаційна модель, комп'ютерне дослідження, оптимізація, пошук шляху, програмування, складське приміщення*

Постановка проблеми. Автоматизоване складське приміщення являє собою сукупність стелажних конструкцій та автоматичних транспортерів, які виконують операції із завантаження та розвантаження збережуваних виробів без участі людини, що дозволяє уникнути помилок у роботі, які зазвичай спричинює людський фактор. До того ж, автоматизація складу сприяє підвищенню пропускної здатності складського приміщення, прискорює всі основні складські процеси, а також дозволяє відстежувати стан і поточні завдання транспортерів та положення усіх збережуваних і транспортованих виробів у реальному часі (для виконання цього завдання використовується інформаційно-керівна система, так звана SCADA-система).

Саме інформаційно-керівна система відповідає за автоматизацію операцій транспортування, гарантує візуалізацію та управління даними щодо кожного об'єкта у

складському приміщенні, здійснює оптимізацію маршруту, часу очікування та логіки переміщення. Система управління, що обслуговує складське приміщення дозволяє максимально ефективно використовувати потенціал складського устаткування, а автоматичні транспортери – максимально оптимізувати обсяги зберігання [1].

Мета статті. Визначення і обрання найефективнішого за техніко-економічними показниками пошукового алгоритму з існуючих шляхом розробки інформаційно-керівної системи, яка імітує роботу реального автоматизованого складу.

Досягнення поставленої мети можливе виконанням таких основних завдань:

1. Розробка та реалізація інформаційно-керівної системи, яка передбачає генерацію складського приміщення з певною конфігурацією (задається кількість рядів, стелажів у ряді і роботів-транспортерів) та здійснення переміщення транспортерів складським приміщення за обраним алгоритмом пошуку найкоротшого шляху.
2. Проведення комп'ютерних досліджень розробленої інформаційно-керівної системи з метою оцінки її функціонування в реальних умовах складських процесів.
3. Проведення експериментальних досліджень розробленої інформаційно-керівної системи з метою оцінки її динамічних характеристик.
4. Виконання техніко-економічного розрахунку для кожного з пошукових алгоритмів та порівняння їх результатів.

Виклад матеріалу. Для виконання дослідження були складені блок-схеми існуючих алгоритмів пошуку найкоротшого шляху: алгоритм «A*» (рис. 1), «пошук у ширину» (рис. 2), «жадібний пошук» (рис. 3), алгоритм Дейкстри (рис. 4) і «Jump Point Search» (рис. 5 – 6). Алгоритм «A*» здійснює пошук по першому найкращому збігу на графі та знаходить маршрут із найменшою вартістю від однієї точки (початкової) до іншої (кінцевої). «Пошук у ширину» працює шляхом послідовного перегляду окремих рівнів графа, починаючи з вузла-джерела; після того як будуть перевірені всі ребра, що виходять із кожного вузла, з черги витягується наступний вузол, і процес повторюється. «Жадібний пошук» є аналогічним алгоритму пошуку шляху «A*», але з тією відмінністю, що, вибираючи нову вершину, він ураховує вартість шляху тільки від попередньої, а не від початкової вершини, як в алгоритмі «A*». Алгоритм Дейкстри покроково перебирає всі вершини графа і призначає їм мітки, які є відомою мінімальною відстанню від вершини джерела до конкретної вершини. Алгоритм «Jump Point Search» є поліпшеним варіантом алгоритму пошуку шляху «A*»; даний алгоритм прискорює пошук шляху за рахунок пропускання багатьох місць пошуку, які переглядаються за звичайної реалізації алгоритму «A*» [2, с. 105 – 120].

В блок-схемах при цьому було прийнято: *start* – поточне положення транспортера; *end* – точка призначення; *grid* – поле пошуку.

Масив доступних пошуковому алгоритму точок – *open*.

node, h, w, n, ns, i, l, s, g, x, ng, orig – довільні змінні.

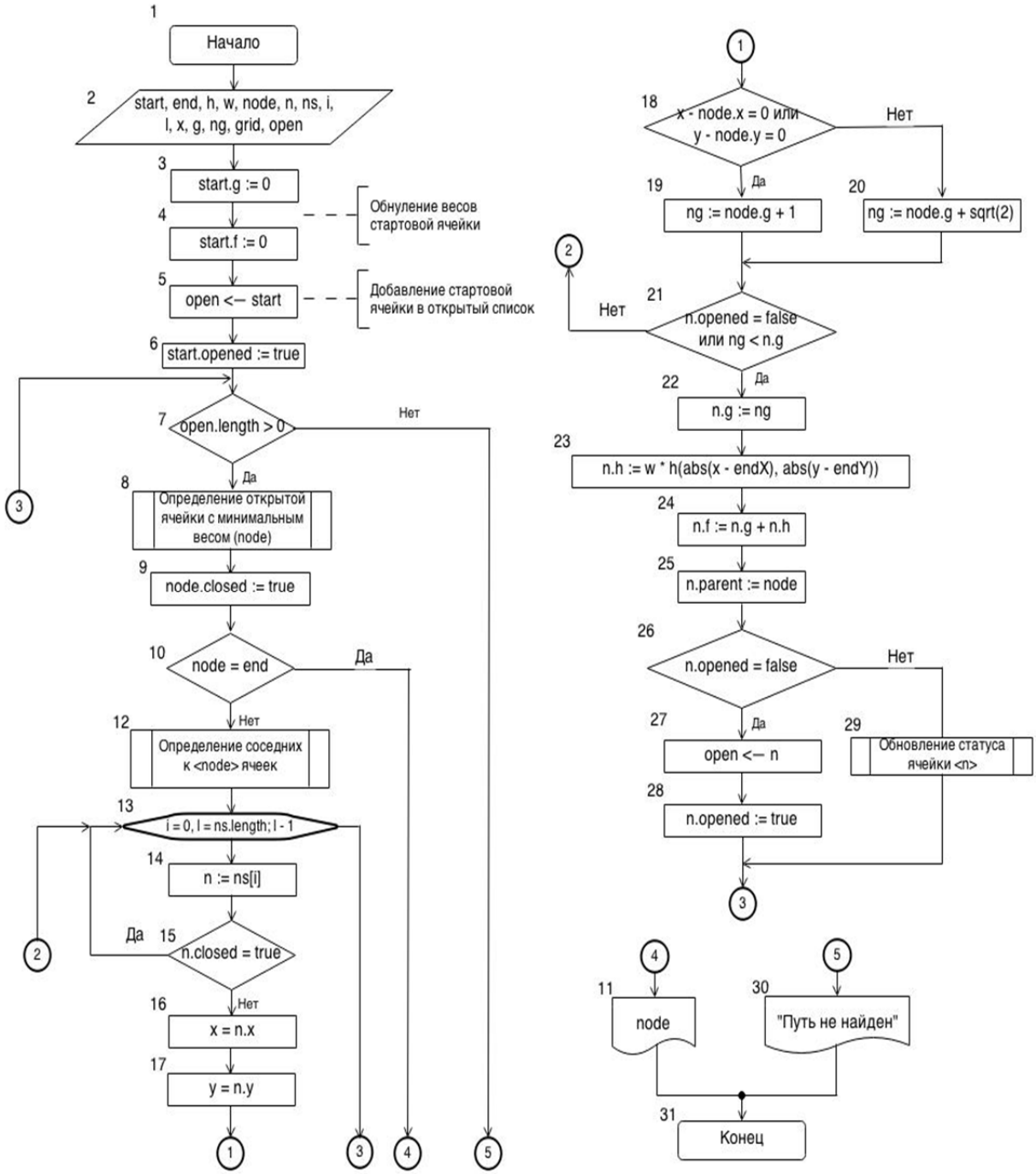

Рис. 1. Алгоритм пошуку найкоротшого шляху «A»*

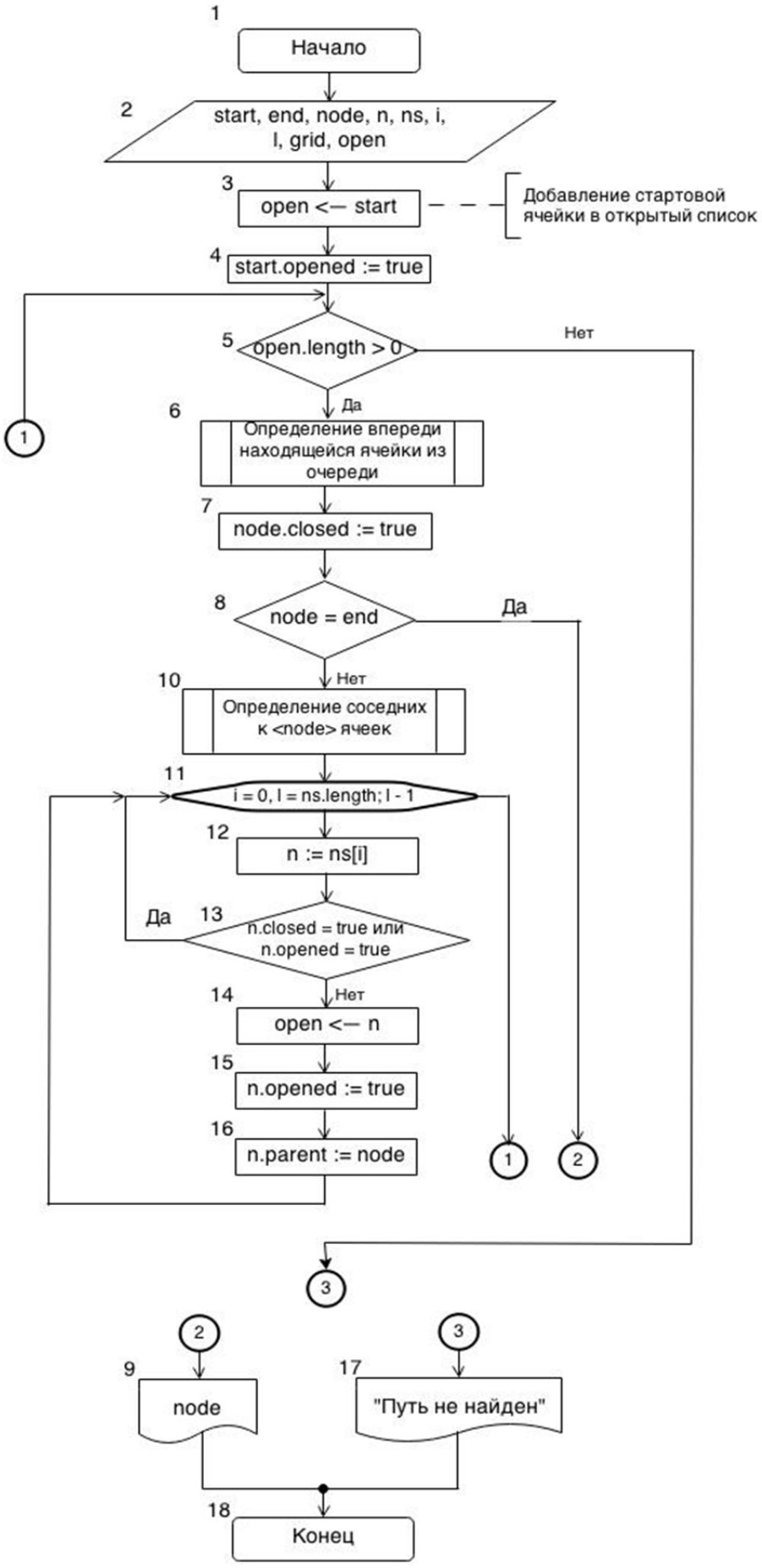

Рис. 2. Алгоритм пошуку найкоротшого шляху «пошук у ширину»

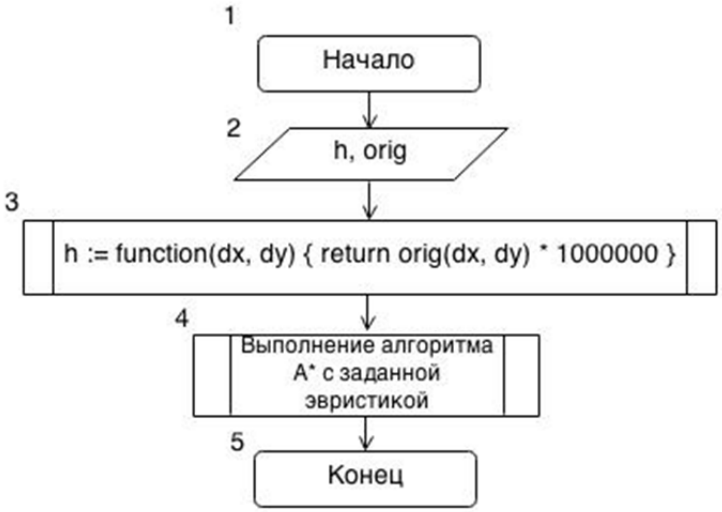

Рис. 3. Алгоритм пошуку найкоротшого шляху «жадібний пошук»

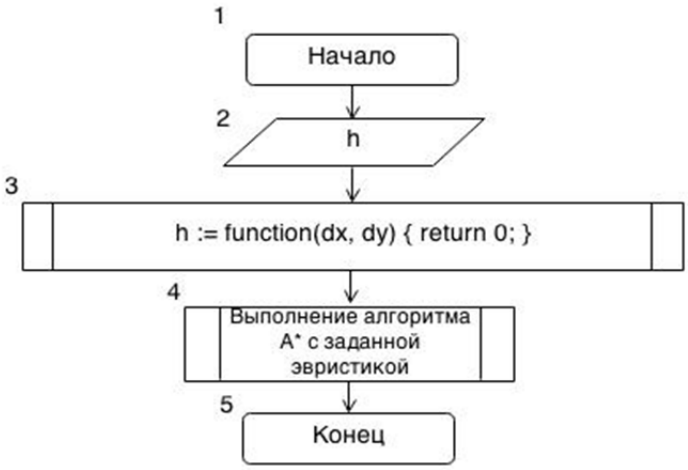

Рис. 4. Алгоритм пошуку найкоротшого шляху Дейкстри

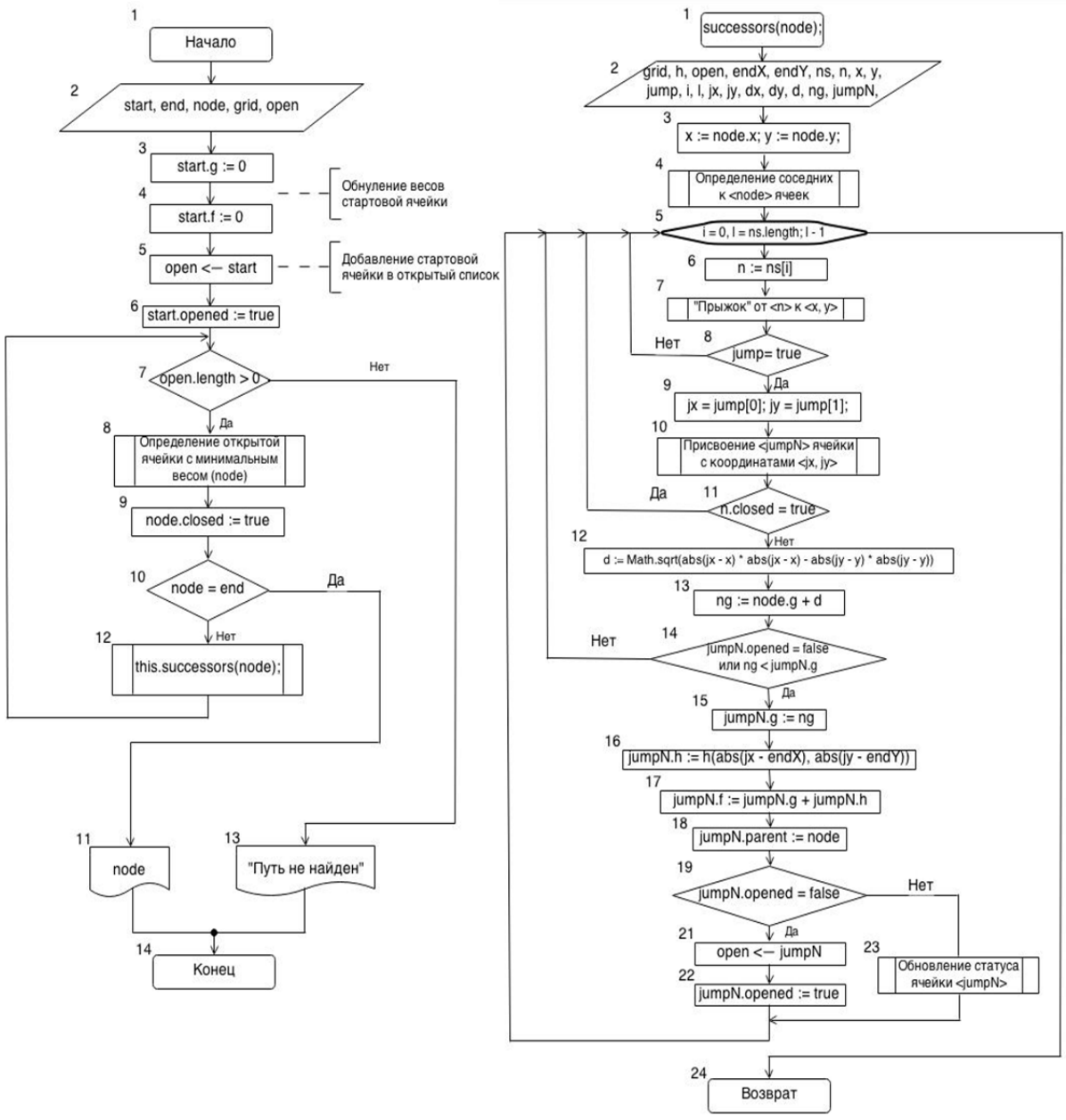

Рис. 5. Алгоритм пошуку найкоротшого шляху «Jump Point Search»

Для програмування та реалізації алгоритмів пошуку найкоротшого шляху використана мова програмування ECMAScript стандарту JavaScript.

Нижче наведено програми реалізації алгоритмів пошуку найкоротшого шляху мовою ECMAScript з бібліотеки PathFinding.js:

```
AStarFinder = function(startX, startY, endX, endY, grid) {
    var open = new Heap(function(nodeA, nodeB) {
        return nodeA.f  nodeB.f;
    }),
    start = grid.getNodeAt(startX, startY),
    end = grid.getNodeAt(endX, endY),
    heuristic = this.heuristic,
    w = this.w, abs = Math.abs, SQRT2 = Math.SQRT2,
    node, ns, n, i, l, x, y, ng;
  start.g = 0;
  start.f = 0;
  open.push(start);
  start.opened = true;
```

```
    while (!open.empty()) {
        node = open.pop();
        node.closed = true;
        if (node === end) {
            return Util.backtrace(end);
        }
        ns = grid.getNeighbors(node);
        for (i = 0, l = ns.length; i < l; ++i) {
            n = ns[i];
            if (n.closed) {
                continue;
            }
            x = n.x;
            y = n.y;
            ng = node.g + ((x - node.x === 0 || y - node.y === 0) ? 1 : SQRT2);
            if (!n.opened || ng < n.g) {
                n.g = ng;
                n.h = n.h || w * heuristic(abs(x - endX), abs(y - endY));
                n.f = n.g + n.h;
                n.parent = node;
                if (!n.opened) {
                    open.push(n);
                    n.opened = true;
                } else {
                    open.updateItem(n);
                }
            }
        }
    }
    return [];
};
BreadthFirstFinder = function(startX, startY, endX, endY, grid) {
    var open = [],
        start = grid.getNodeAt(startX, startY),
        end = grid.getNodeAt(endX, endY),
        ns, n, node, i, l;
    open.push(start);
    start.opened = true;
    while (open.length) {
        node = open.shift();
        node.closed = true;
        if (node === end) {
            return Util.backtrace(end);
        }
        ns = grid.getNeighbors(node);
        for (i = 0, l = ns.length; i < l; ++i) {
            n = ns[i];
            if (n.closed || n.opened) {
                continue;
            }
            open.push(n);
            n.opened = true;
            n.parent = node;
        }
    }
    return [];
};
```

```
BestFirstFinder  = function(opt) {
    AStarFinder(this, opt);
    var orig = this.heuristic;
    this.heuristic = function(dx, dy) {
        return orig(dx, dy) * 1000000;
    };
};
DijkstraFinder  = function(opt) {
    AStarFinder(this, opt);
    this.heuristic = function(dx, dy) {
        return 0;
    };
}
JumpPointFinder = function(startX, startY, endX, endY, grid) {
    var open = this.open = new Heap(function(nodeA, nodeB) {
            return nodeA.f - nodeB.f;
        }),
        start = this.start = grid.getNodeAt(startX, startY),
        end = this.end = grid.getNodeAt(endX, endY), node;
    this.grid = grid;
    start.g = 0;
    start.f = 0;
    open.push(start);
    start.opened = true;
    while (!open.empty()) {
        node = open.pop();
        node.closed = true;
        if (node === end) {
            return Util.expandPath(Util.backtrace(end));
        }
        this._successors(node);
    }
    return [];
};
JumpPointFinder.prototype._successors = function(node) {
    var grid = this.grid,
        h = this.heuristic,
        open = this.open,
        endX = this.end.x,
        endY = this.end.y,
        ns, n, jump, i, l,
        x = node.x, y = node.y,
        jx, jy, dx, dy, d, ng, jumpN,
        abs = Math.abs, max = Math.max;
    ns = this._findNeighbors(node);
    for(i = 0, l = ns.length; i < l; ++i) {
        n = ns[i];
        jump = this._jump(n[0], n[1], x, y);
        if (jump) {
            jx = jump[0];
            jy = jump[1];
            jumpN = grid.getNodeAt(jx, jy);
            if (jumpN.closed) {
                continue;
            }
            d = Math.sqrt(abs(jx - x) * abs(jx - x) - abs(jy - y) * abs(jy - y));
            ng = node.g + d; // следующее значение веса `g`
```

```
                if (!jumpN.opened || ng < jumpN.g) {
                    jumpN.g = ng;
                    jumpN.h = h(abs(jx - endX), abs(jy - endY));
                    jumpN.f = jumpN.g + jumpN.h;
                    jumpN.parent = node;
                    if (!jumpN.opened) {
                        open.push(jumpN);
                        jumpN.opened = true;
                    } else {
                        open.updateItem(jumpN);
                    }
                }
            }
        }
    };
```

Спроектовані та розроблені програми мовами Delphi, ECMAScript та PHP дозволяють провести повноцінні дослідження імітаційної моделі інформаційно-керівної системи складського приміщення.

Імітаційна модель була створена мовою ECMAScript із графічною бібліотекою Pixi.js і працює без використання допоміжних програм, сама при цьому будучи незалежною комп'ютерною програмою. На рисунку 6 зображено імітаційну модель інформаційно-керівної системи складського приміщення під час її роботи в згенерованому програмою типовому приміщенні розмірністю 4 ряди, кожен з яких вміщує 17 пар стелажів та яке обслуговується 4-ма роботами-транспортерами. Дана модель дозволяє: імітувати завантаження виробу (до найближчого стелажу від місця його прийняття, до найближчого стелажу від зони розвантаження, або ручне завантаження до вказаного користувачем незайнятого стелажу) та імітувати відпуск виробу (найближчий до зони розвантаження, найвіддаленіший від зони розвантаження, або ручний відпуск зі вказаного користувачем зайнятого стелажу). Як процедура завантаження, так і процедура відпуску можуть бути встановлені в неперервний режим роботи, тобто до повного завантаження або розвантаження усього складу відповідно.

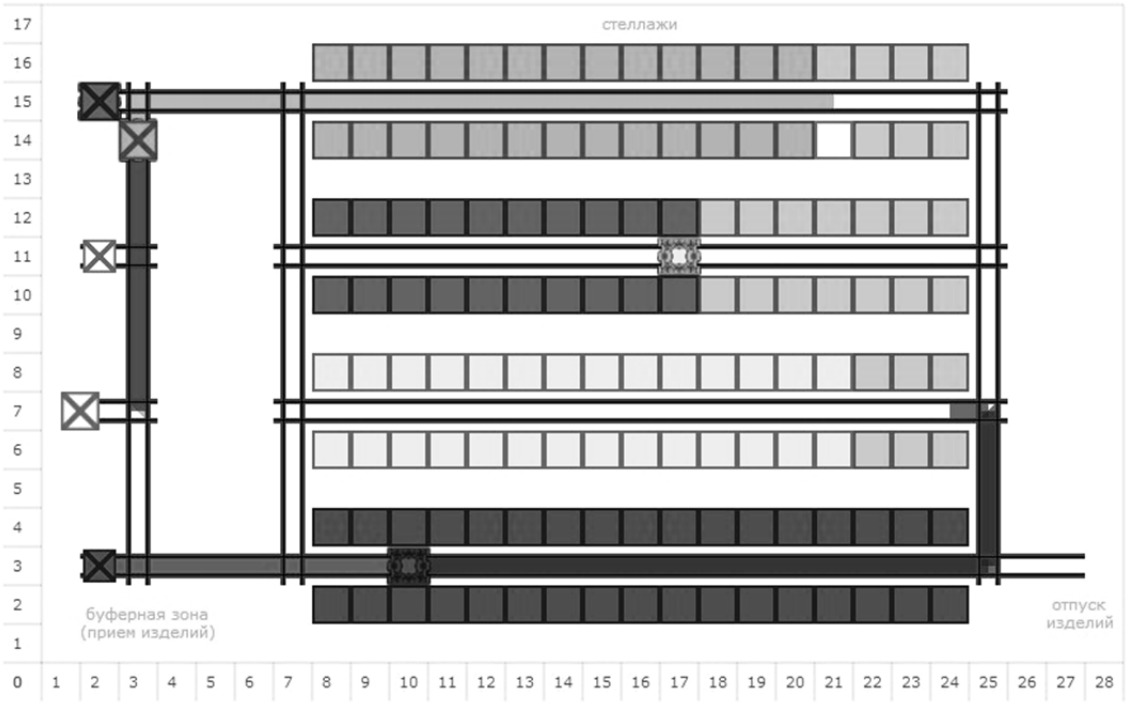

Рис. 6. Модель інформаційно-керівної системи складу для приміщення розмірністю 4 × 34

Також у системі передбачено вибір із запропонованих алгоритму роботи транспортерів, згідно з яким працює функція пошуку найкоротшого шляху. Причому вибір може бути здійснений у будь-який момент часу і роботи-транспортери відразу ж перепрограмовуються під новий пошуковий алгоритм і починають працювати згідно з ним.

Для більш точного відтворення реальної інформаційно-керівної системи складського приміщення була реалізована спеціальна база даних, що містить повну інформацію про кожний із роботів-транспортерів (ідентифікаційний номер у системі, мережна IP-адреса, поточне завдання, координатне положення та деякі статистичні дані) та стелажів (ідентифікаційний номер у системі, поточний статус, координатне положення та статистичні дані). Усі дані оновлюються в реальному часі.

Для визначення найефективнішого пошукового алгоритму були проведені імітаційні експерименти повного завантаження порожнього та розвантаження зайнятого складського приміщення для кожного з алгоритмів, із визначенням сумарного часу роботи кожного з них, та кількості розрахункових операцій, які були виконані під час їх роботи.

У довільному порядку були вибрані конфігурації складу з кількістю рядів: 1, 2 та 4 і кількістю стелажів у кожному: 10, 26 та 40.

Результуючі сумарні значення часу та кількості операцій для кожного алгоритму зображені у вигляді діаграм на рисунку 8:

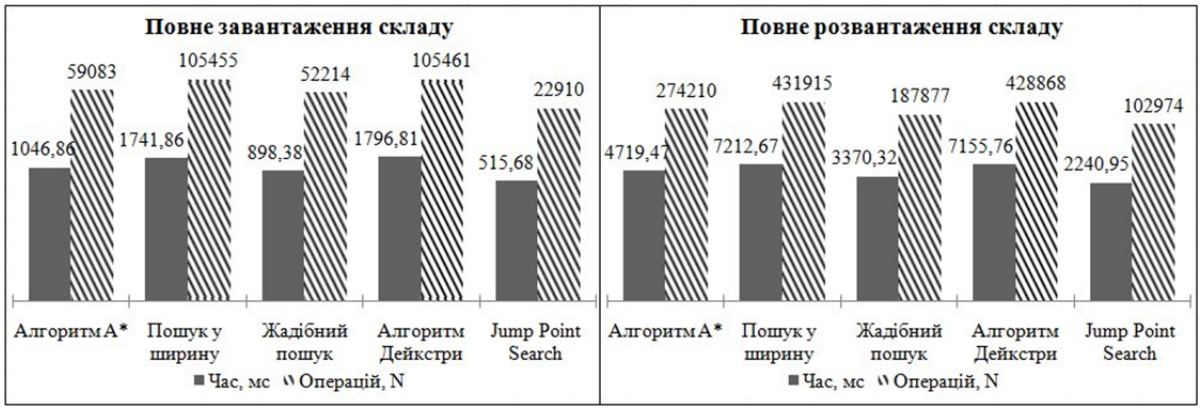

Рис. 8. Підсумкові результати імітаційного експерименту

Детальні результати проведеної імітації на повне завантаження та повне розвантаження наведені у таблицях 1 та 2 відповідно, де t – час роботи алгоритму у мілісекундах, N – кількість розрахункових операцій.

Найефективніші показники часу роботи та кількості розрахункових операцій незалежно від вибраної конфігурації складського приміщення отримані із застосуванням сучасного (розроблений у 2011 році) алгоритму «Jump Point Search». Причому з ростом кількості обслуговуваних стелажів перевага цього алгоритму перед будь-яким з інших стає ще сильнішою, що дозволяє вигравати до 1,2 секунди комп'ютерного часу за повного завантаження, та до 5,0 секунд за повного розвантаження. Використання даного алгоритму сприятиме не тільки скороченню часу застосування ресурсів обчислювальної техніки, а й зниженню витрат на її утримання за рахунок зменшення кількості розрахункових операцій, тобто оптимізації використання обчислювальної техніки в цілому.

Таблиця 1

Результати імітаційного експерименту на повне завантаження складу

№	Пошуковий алгоритм	Кількість рядів																		**∑(t), мс**	**∑(N)**
		1						2						4							
		Стелажів						Стелажів						Стелажів							
		10		26		40		10		26		40		10		26		40			
		t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N		
1	Алг. А*	11,17	334	22,69	1286	41,63	2540	16,57	953	54,57	3397	128,93	7142	67,53	4203	247,74	13946	456,03	25282	1046,86	59083
2	«У ширину»	7,46	422	25,86	1636	52,74	3224	29,30	1876	106,68	6026	192,84	11236	108,37	6815	437,27	26867	781,34	47353	1741,86	105455
3	«Жадібний»	9,28	340	22,87	1300	39,88	2560	13,46	880	56,01	3357	95,81	5720	64,05	3770	209,59	12450	387,43	21837	898,38	52214
4	Дейкстри	6,43	420	26,88	1636	50,72	3224	27,34	1811	97,31	6133	218,38	11415	119,33	7286	411,80	25088	838,62	48448	1796,81	105461
5	Jump P.S.	**2,37**	**80**	**9,64**	**208**	**7,59**	**320**	**13,07**	**446**	**28,26**	**1356**	**48,96**	**2104**	**48,74**	**2408**	**120,78**	**6152**	**236,27**	**9836**	**515,68**	**22910**

Таблиця 2

Результати імітаційного експерименту на повне розвантаження складу

№	Пошуковий алгоритм	Кількість рядів																		**∑(t), мс**	**∑(N)**
		1						2						4							
		Стелажів						Стелажів						Стелажів							
		10		26		40		10		26		40		10		26		40			
		t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N	t,мс	N		
1	Алг. А*	10,99	681	49,17	3182	131,15	8123	55,19	3264	205,13	12098	535,82	31375	183,79	11313	1159,89	65453	2388,34	138721	4719,47	274210
2	«У ширину»	16,35	590	56,94	3432	139,56	8555	54,25	3353	261,62	16501	714,12	42665	288,77	18763	2014,57	116792	3666,49	221264	7212,67	431915
3	«Жадібний»	9,45	528	53,98	3162	135,16	8129	31,97	2066	156,89	9735	424,58	24938	144,66	8874	864,09	45611	1549,54	84834	3370,32	187877
4	Дейкстри	10,29	587	54,16	3452	134,23	8535	55,11	3347	256,25	16215	722,45	43024	303,22	19537	2007,48	117624	3612,57	216547	7155,76	428868
5	Jump P. S.	**4,84**	**222**	**17,58**	**724**	**27,39**	**1288**	**28,88**	**1724**	**104,10**	**5662**	**255,12**	**11700**	**134,11**	**7556**	**647,19**	**27890**	**1021,74**	**46208**	**2240,95**	**102974**

Висновки. 1. Реалізовано програми алгоритмів пошуку найкоротшого шляху мовою програмування ECMAScript стандарту JavaScript.

2. Розроблено та реалізовано імітаційну модель інформаційно-керівної системи складського приміщення мовою ECMAScript з графічною бібліотекою Pixi.js.

3. Проведено комп'ютерні дослідження розробленої моделі інформаційно-керівної системи з оцінкою її функціонування в реальних умовах складських процесів.

4. Порівняльним розрахунком роботи кожного з пошукових алгоритмів визначено найефективніший із техніко-економічних міркувань – «Jump Point Search», перевага в часі роботи якого перед іншими становить від 51 до 248 %, а в кількості розрахункових операцій – від 82 до 360 %.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. **Norbert Ascheuer, Martin Grotschel, Atef Abdel-Aziz Abdel-Hamid**. Order Picking in an Automatic Warehouse: Solving Online Asymmetric TSPs. – Berlin : Konrad-Zuse-Zentrum fűr Informationstechnik Berlin, 1998. – 15 с. – (Препринт / Konrad-Zuse-Zentrum Berlin; SC 98-08).

2. **Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani**. Algorithms. – McGraw-Hill, 2006.

3. AMSEL – A Modelling and Simulation Environment Library. Developed at the Konrad-Zuse-Zentrum fËur Informationstechnik Berlin [Електронний ресурс] // ZIB, 1997. – Режим доступу: http://www.zib.de/ascheuer/AMSEL.

4. **Ascheuer N.** Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems. – Berlin : Technical Univiversity Berlin, 1995.

SUMMARY

Problem statement. Automatic warehouse is a collection of shelving constructions and robotic transporters, performing operations of loading and unloading stored articles without the participation of human, that allows to avoid errors in the processing, which may result of human factors.

Research objective. The aim of the study is to determine and accept the most efficient (by technical and economic performance) pathfinding algorithm from existing by developing warehouse control system, which simulates the real processes in the automatic warehouse. Achieving this goal is possible by perform the following major tasks:

1. Development and implementation of warehouse control system that provides generation of a warehouse model with a specific configuration (specified number of rows, shelves in a row and robotic transporters) and the realization of robotic transporters moving in the warehouses by chosen pathfinding algorithm finding the shortest path.
2. Carrying out computer studies of developed model of the warehouse control system with the evaluation of its performance in the real automatic warehouse processes.
3. Performing experimental research of the developed warehouse control system to evaluate its dynamic characteristics.
4. Performing feasibility calculations for each of the pathfinding algorithms and comparing their results.

Conclusions. As a result, are realized programs of pathfinding algorithms on ECMAScript programming language using JavaScript standard. Developed and implemented a simulation model of warehouse control system on ECMAScript language with graphics library Pixi.js. Were carried out computer studies of developed model of the warehouse control system with the evaluation of its performance. Performed comparative calculations of each pathfinding algorithm with determining the most effecient of them.

REFERENCES

1. **Norbert Ascheuer, Martin Grotschel, Atef Abdel-Aziz Abdel-Hamid**. Order Picking in an Automatic Warehouse: Solving Online Asymmetric TSPs. – Berlin: Konrad-Zuse-Zentrum fűr Informationstechnik Berlin, 1998. – 15 s. – (Preprint / Konrad-Zuse-Zentrum Berlin; SC 98-08).

2. **Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani**. Algorithms. – McGraw-Hill, 2006.

3. AMSEL – A Modelling and Simulation Environment Library. Developed at the Konrad-Zuse-Zentrum fËur Informationstechnik Berlin [Elektronnyy resurs] // ZIB, 1997. – Rezhym dostupu: http://www.zib.de/ascheuer/AMSEL.

4. **Ascheuer N.** Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems. – Berlin: Technical Univiversity Berlin, 1995.