ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра ЕОМ

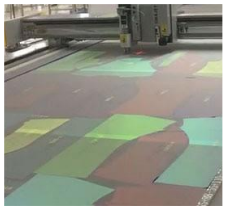
Кваліфікаційна робота
на тему:

Методи вирішення задачі прямокутного гільйотинного розкрою листового матеріалу

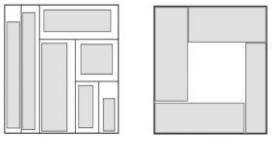
Здобувач:
Анастасія КОНОНЕНКО
СПм-23-4

Керівник:
Георгій ІВАЩЕНКО
доц. каф. ЕОМ

АКТУАЛЬНІСТЬ ПРОБЛЕМИ

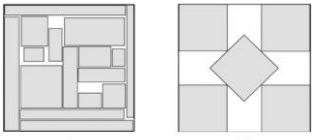


- Текстиль
- Меблі
- Будівництво
- Поліграфія



а) б)

Двомірний прямокутний: а) гільйотинний розкрій; б) негільйотинний розкрій



а) б)

Двомірний прямокутний: а) ортогональний розкрій; б) неортогональний розкрій

ЗАДАЧА ГІЛЬЙОТИННОГО РОЗКРОЮ

Мінімізація витрат матеріалу безпосередньо залежить від максимізації КВМ, оскільки цей показник відображає ефективність використання матеріалу.

Таким чином цільова функція максимізації (1) має наступний вигляд : $\frac{\sum_{i=1}^{N}(w_i' \times h_i')}{M \times W \times H} \to max.$ (1)

Обмеження задачі включають:

$0 \le x_i + w_i \le W, 0 \le y_i + h_i \le H, \forall i,$ (2)

$(x_i + w_i \le x_j) \vee (x_j + w_j \le x_i) \vee (y_i + h_i \le y_j) \vee (y_j + h_j \le y), \forall i \ne j,$ (3)

$\sum_{i=1}^{N}(w_i' \times h_i') \le M \times W \times H,$ (4)

$w_i' = \begin{cases} w_i, \text{якщо } r_i = 0, \\ h_i, \text{якщо } r_i = 1, \end{cases}$ (5)

$h_i' = \begin{cases} h_i, \text{якщо } r_i = 0, \\ w_i, \text{якщо } r_i = 1, \end{cases}$ (6)

$v_l \in \{0, W\} \cup \{v_{l'} \mid l' < l\}, h_l \in \{0, H\} \cup \{h_{l'} \mid l' < l\},$ (7)

$v_l \ne x_i, v_l \ne x_i + w_i, \forall i, h_l \ne y_i, h_l \ne y_i + h_i, \forall i,$ (8)

де $W \times H$ – ширина та висота аркуша,

$w_i \times h_i$ – ширина та висота i-го елемента на аркуші ($i = 1,2, \dots, N$),

x_i, y_i – координати лівого верхнього кута i-го елемента на аркуші ($i = 1,2, \dots, N$),

M – кількість використаних аркушів,

$r_i \in \{0, 1\}$ – ознака повороту i-го елемента,

v_l, h_l – координати вертикальних і горизонтальних розрізів,

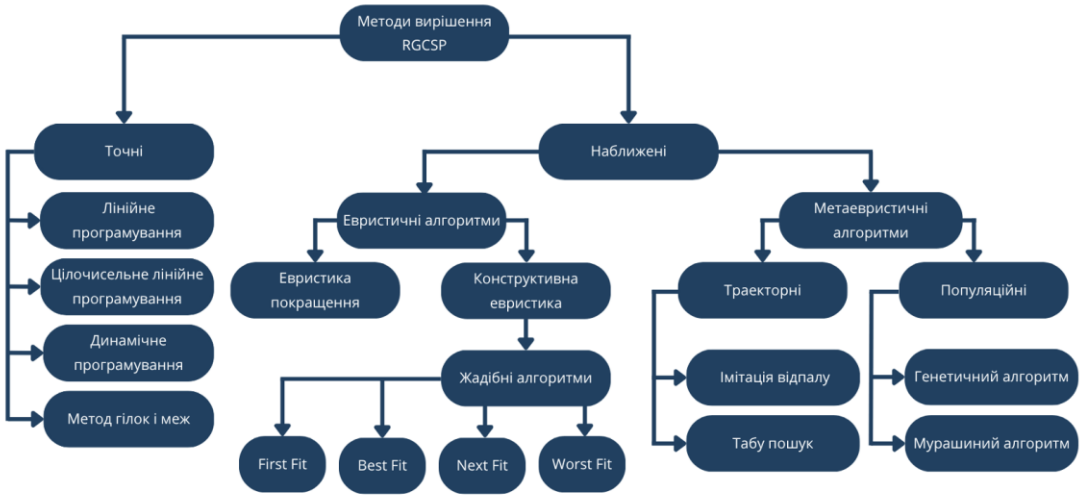
l – індекс розрізу.

ПОСТАНОВКА ЗАДАЧІ

Метою роботи є проведення аналізу наближених алгоритмів рішення задачі прямокутного гільйотинного розкрою на тестових даних різних розмірів. Для дослідження роботи алгоритмів необхідно виконати такі завдання:

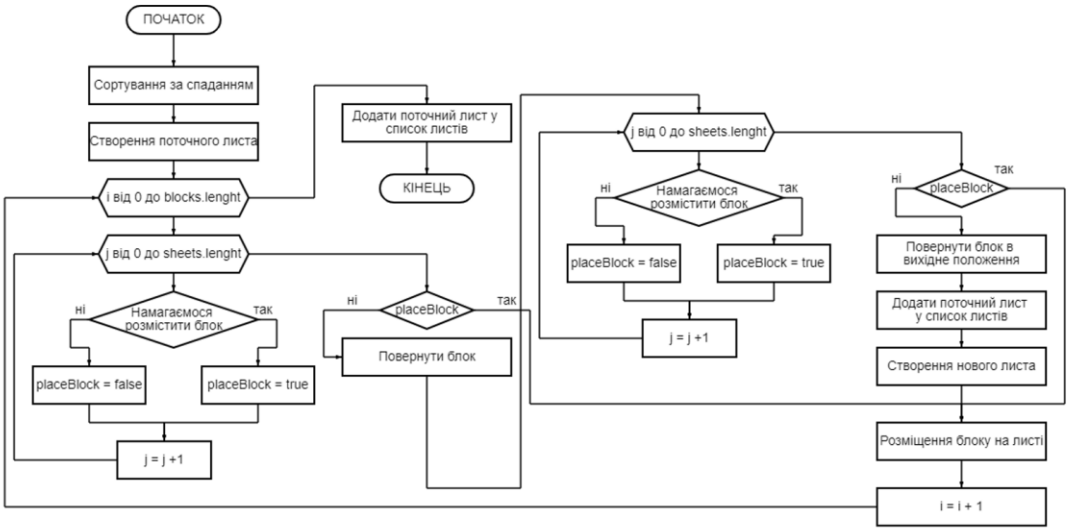
- провести аналіз актуальних наукових досліджень, присвячених вирішенню задач гільйотинного розкрою;
- обрати та проаналізувати поширені наближені алгоритми для рішення RGCSP;
- розробити гібридний алгоритм на основі існуючих методів;
- визначити необхідні значення параметрів для обраних алгоритмів;
- забезпечити можливість запускати серії алгоритмів для вирішення наборів RGCSP, із можливістю перегляду, аналізу та порівняння отриманих результатів;
- забезпечити можливість візуалізації результатів роботи алгоритмів;
- провести порівняльний аналіз роботи реалізованих алгоритмів.

ІСНУЮЧІ МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ РОЗКРОЮ



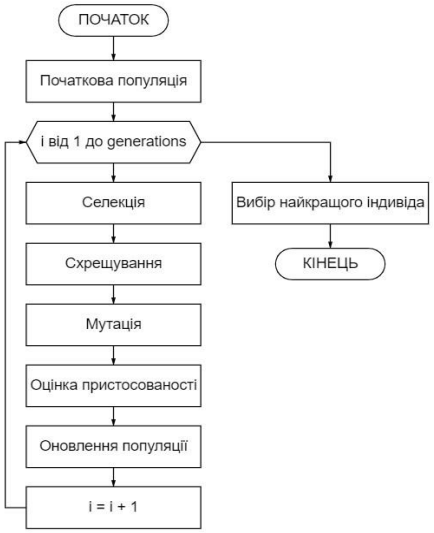
5

АЛГОРИТМ BFD



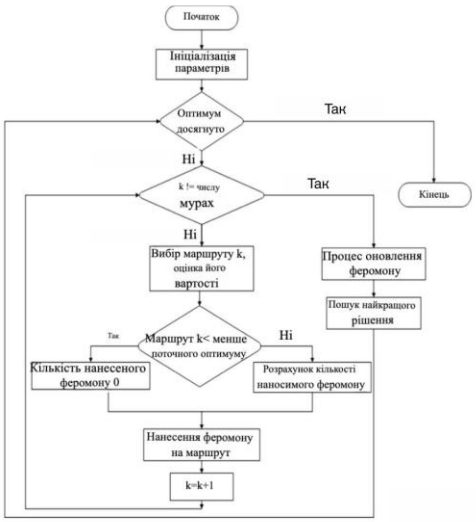
6

ГЕНЕТИЧНИЙ АЛГОРИТМ



- Початкова популяція – випадкові координати блоків.
- Селекція – турнірна селекція випадкових особин з популяції для вибору батьків.
- Схрещування (One-point, Arithmetic, OX, PMX) – комбінування генів батьків для створення нащадків.
- Мутація – випадкова зміна координат блоку, обмін блоками місцями, випадкова зміна орієнтації блоку та перенесення блоку між листами.
- Оцінка пристосованості – оцінка компактності блоків при розташуванні.

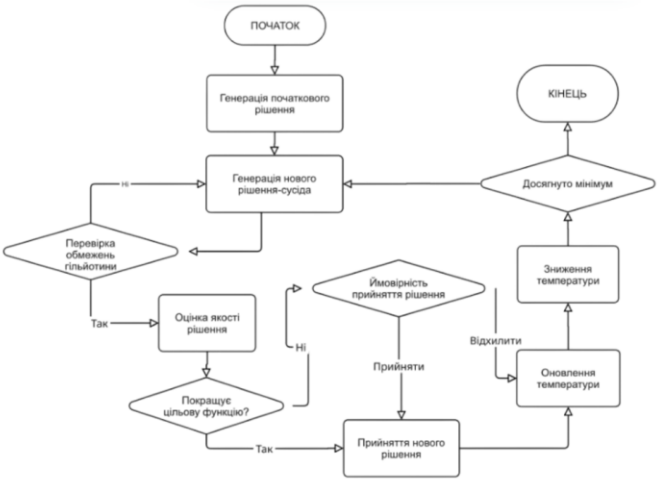
АЛГОРИТМ МУРАШИНОЇ КОЛОНІЇ



$$P_i = \frac{\tau_i^\alpha \eta_i^\beta}{\sum_{k=0}^{N} \tau_k^\alpha \eta_k^\beta} \qquad (9)$$

де, P_i – ймовірність переходу шляхом і;
τ – кількість феромону на шляху;
η – величина зворотна вартості переміщення;
α – коефіцієнт, що регулює вплив феромонів;
β – коефіцієнт, що регулює вплив евристичної інформації.

АЛГОРИТМ СИМУЛЯЦІЇ ВІДПАЛУ



Шанс прийняття нового рішення визначається за формулою (10):

$$P_{s+1} = e^{-\Delta/T} \qquad (10)$$

де P_{s+1} – шанс прийняття нового рішення;

Δ – різниця вартості рішень між новим результатом та попереднім;

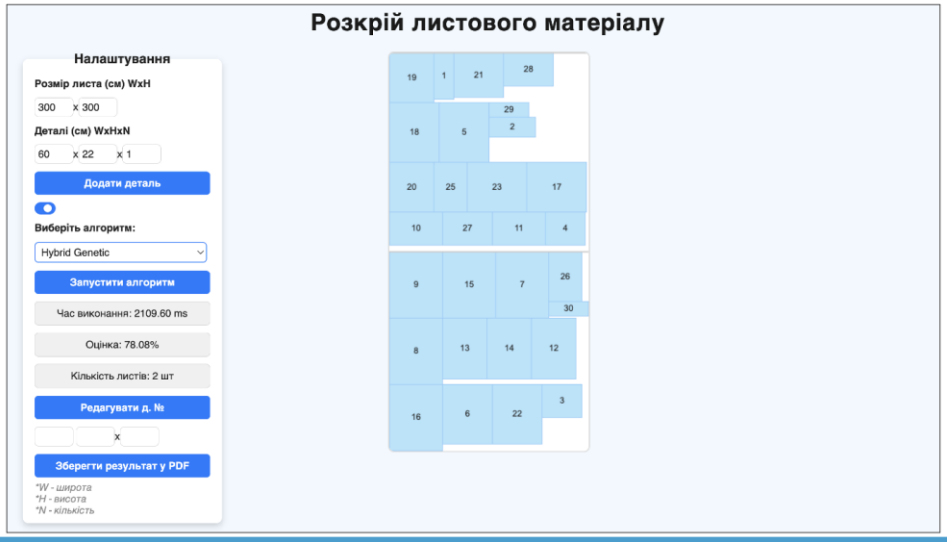
T – «температура» стану рішення.

СТЕК ТЕХОЛОГІЙ

HTML JS CSS CANVAS

jQuery jsPDF LocalStorage GitHub

ІНТЕРФЕЙС ТЕСТОВОЇ ПРОГРАМИ



АНАЛІЗ ВПЛИВУ ПАРАМЕТРІВ ГА

Кількість ітерацій	Критерій ефективності	Кількість деталей				
		20	40	60	80	100
1000	Час, мс	301.31	791.69	1405.65	2070.48	3091.61
	Оцінка	40	36.94	47.39	53.39	43.43
	Кількість листів	1	2	2	2	3
2000	Час, мс	661,07	1545.59	2944.15	4432.73	6556.42
	Оцінка	44,45	41.48	46.20	53.96	43.69
	Кількість листів	1	1,8	2	2	3
3000	Час, мс	958.73	2586.94	4543.96	6835.71	37500,4
	Оцінка	41.96	47.13	41.51	54.61	45,45
	Кількість листів	1	1,6	2	2	3
4000	Час, мс	1435.77	3244.86	6238.01	9408.26	13321.07
	Оцінка	48.47	39.67	44.03	54.71	48.10
	Кількість листів	1	1.80	2	2	2,8
5000	Час, мс	1605.97	4064.81	7885.90	12011.47	16882.59
	Оцінка	49,68	49.47	41.28	56.87	52.00
	Кількість листів	1	1,6	2	2	2,6

Кроссовер	Критерій ефективності	Кількість деталей				
		20	40	60	80	100
One-point	Час, мс	1559.57	4183.73	7898.16	11983.36	17616.3
	Оцінка	44.49	40.35	45.46	54.10	48.53
	Кількість листів	1	2	2	2	2.80
Arithmetic	Час, мс	1655.18	3318.82	5206.13	6981.40	10472.07
	Оцінка	46.53	44.44	44.64	54.70	43.74
	Кількість листів	1	1,8	2	2	3
OX	Час, мс	2846.50	9227.03	12695.42	19661.12	31994.69
	Оцінка	46.17	40.83	43.12	54.48	51.91
	Кількість листів	1	1,8	2	2	2.60
PMX	Час, мс	22633.05	44803.65	85396.8	125915.97	180799.42
	Оцінка	46.38	47.83	46.51	55.87	44.27
	Кількість листів	1	1.60	2	2	3

ВПЛИВ ПАРАМЕТРІВ ACO

Кількість ітерацій	Критерій ефективності	Кількість деталей				
		20	40	60	80	100
1000	Час, мс	1663.13	6047.98	13505.43	23031.00	33386.59
	Оцінка	40.63	64.87	44.30	55.30	44.28
	Кількість листів	1	1	2	2	3
2000	Час, мс	1601.92	5953.99	13694.91	22513.37	33245.01
	Оцінка	42.08	64.95	42.75	55.06	44.26
	Кількість листів	1	1	2	2	3
3000	Час, мс	1603.02	5952.39	13622.83	23414.68	33229.24
	Оцінка	48.32	65.62	40.83	53.90	42.43
	Кількість листів	1	1	2	2	3
4000	Час, мс	1591.90	13505.43	13741.65	23364.34	33304.91
	Оцінка	43.81	44.30	41.39	53.88	43.13
	Кількість листів	1	2	2	2	3

Кількість мурах	Критерій ефективності	Кількість деталей				
		20	40	60	80	100
100	Час, мс	1633.71	6092.32	13459.86	23720.86	32057.98
	Оцінка	48.99	65.02	44.30	54.30	45.25
	Кількість листів	1	1	2	2	3
300	Час, мс	3877.79	16217.44	36892.30	60343.98	125928.61
	Оцінка	49.92	66.69	43.10	53.11	48.75
	Кількість листів	1	1	2	2	2.8
500	Час, мс	5998.74	24035.86	57389.28	92579.43	183357.19
	Оцінка	42.18	65.03	43.44	54.28	52.30
	Кількість листів	1	1	2	2	2.6
1000	Час, мс	11376.63	43255.34	105432.58	159512.42	250180.50
	Оцінка	37.85	61.42	40.55	53.00	47.67
	Кількість листів	1	1	2	2	2.80

РЕЗУЛЬТАТИ ТЕСТІВ

Кількість деталей	Критерій ефективності	Алгоритм				
		BFD	Генетичний	Мурашина колонія	Симуляція відпалу	Гібридний
20	Час, мс	0.12	1655.18	3877.79	01.02	1474.87
	Оцінка	59.28	46.53	49.92	48.68	65.79
	Кількість листів	1	1	1	1	1
40	Час, мс	0.08	3318.82	16217.44	0.31	2703.93
	Оцінка	65.40	44.44	66.69	37.13	64.08
	Кількість листів	1	1,8	1	2	1
60	Час, мс	0.06	5206.13	36892.30	0.53	4235.27
	Оцінка	42.18	44.64	43.10	45.53	42.18
	Кількість листів	2	2	2	2	2
80	Час, мс	0.12	6981.40	60343.98	0.59	5505.68
	Оцінка	57.36	54.70	53.11	35.99	56.21
	Кількість листів	2	2	2	3	2
100	Час, мс	0.15	10472.07	125928.61	0.72	6907.31
	Оцінка	45.18	43.74	48.75	43.37	49.58
	Кількість листів	3	2.8	2.8	3	2.8

Налаштування **генетичного алгоритму** :

- кількість ітерацій – 5000;
- тип кросоверу – Arithmetic;
- метод селекції – турнірний відбір (2 учасники);
- початкова популяція – випадкова;
- ймовірність мутації – рівномірно розподілена між чотирма типами мутацій (по 0.25);
- умова зупинки – 200 ітерацій без покращення або досягнення граничної кількості ітерацій.

Налаштування алгоритму **мурашиної колонії** (ACO):

- кількість ітерацій – 1000;
- кількість мурах – 300;
- коефіцієнт випаровування феромону – 0.5;
- $\alpha = 1, \beta = 2$;
- умова зупинки – 200 ітерацій без покращення або досягнення граничної кількості ітерацій.

Алгоритм **імітації відпалу**:

- кількість ітерацій – 1000;
- початкова температура – 100;
- кінцева температура = 1e-4;
- параметр alpha – 0.98.

Гібридний генетичний алгоритм використовує у якості початкової популяції рішення BFD.

BFD без параметрів налаштування так як має фіксовану логіку.

ВИСНОВКИ

У ході роботи було досліджено сучасні алгоритми для задачі гільйотинного розкрою листового матеріалу. Досліджено ефективність п’яти підходів: BFD, ГА, ACO, методу імітації відпалу та гібридний. Для практичної реалізації було створено програмне забезпечення, що автоматизує процес розкрою, дозволяє редагувати вхідні дані, обирати метод, візуалізувати результат і зберігати його.

Результати експериментів показали, що алгоритм BFD демонструє найвищу швидкодію, проте поступається за якістю розміщення. Алгоритм мурашиної колонії забезпечує найвищу якість рішень, однак має суттєві часові витрати, що обмежує його застосування на практиці. Генетичний алгоритм і метод імітації відпалу показали задовільні результати, проте їх стабільність та ефективність варіюються залежно від складності задачі. Найкращі показники збалансованості між швидкістю обчислень та якістю рішень продемонстрував гібридний алгоритм, який виявився найстабільнішим у різних сценаріях.

В процесі роботи опублікована стаття на тему «Метаевристичні методи вирішення задачі гільйотинного розкрою» у журналі «Системи управління, навігації та зв’язку», №1(79).

ДОДАТОК Б

Вихідний код застосунку

Б.1 Реалізація алгоритмів розкрою

Б.1.1 geneticAlgorithm.js

```
function runGeneticAlgorithm(populationSize, mutationRate,
generations, blocks, sheetWidth, sheetHeight, initialPopulation,
crossoverType) {
    function getRandomNumber(min, max) {
        return Math.random() * (max - min) + min;
    }
    function areRectanglesIntersecting(rect1, rect2) {
        let rect1Width, rect1Height, rect2Width, rect2Height;
        if (rect1.orientation === "horizontal") {
            rect1Width = rect1.h;
            rect1Height = rect1.w;
        } else {
            rect1Width = rect1.w;
            rect1Height = rect1.h;
        }
        if (rect2.orientation === "horizontal") {
            rect2Width = rect2.h;
            rect2Height = rect2.w;
        } else {
            rect2Width = rect2.w;
            rect2Height = rect2.h;
        }
        if (
            rect1.x + rect1Width <= rect2.x ||
            rect2.x + rect2Width <= rect1.x ||
            rect1.y + rect1Height <= rect2.y ||
            rect2.y + rect2Height <= rect1.y
        ) {
            return false;
        }
        return true;
    }

    function isGuillotine(blocks, sheetWidth, sheetHeight) {
        let areas = [{ x: 0, y: 0, w: sheetWidth, h: sheetHeight
}];
        for (let i = 0; i < blocks.length; i++) {
            let block = blocks[i];
            let w = block.orientation === "vertical" ? block.w :
```

```
block.h;
            let h = block.orientation === "vertical" ? block.h :
block.w;
            let bx = block.x, by = block.y;
            let found = false;
            for (let j = 0; j < areas.length; j++) {
                let area = areas[j];
                if (
                    bx === area.x && by === area.y &&
                    w <= area.w && h <= area.h
                ) {
                    let ok = true;
                    for (let k = 0; k < i; k++) {
                        let prev = blocks[k];
                        if (areRectanglesIntersecting(block,
prev)) {
                            ok = false;
                            break;
                        }
                    }
                    if (!ok) continue;
                    let newAreas = [];
                    if (area.w - w > 0) {
                        newAreas.push({ x: area.x + w, y:
area.y, w: area.w - w, h: h });
                    }
                    if (area.h - h > 0) {
                        newAreas.push({ x: area.x, y: area.y +
h, w: area.w, h: area.h - h });
                    }
                    for (let k = 0; k < areas.length; k++) {
                        if (k !== j) newAreas.push(areas[k]);
                    }
                    areas = newAreas.filter(a => a.w > 0 && a.h
> 0);
                    found = true;
                    break;
                }
            }
            if (!found) return false;
        }
        return true;
    }

    function getFreeGuillotineAreas(blocks, sheetWidth,
sheetHeight) {
        let areas = [{ x: 0, y: 0, w: sheetWidth, h: sheetHeight
}];
        for (const block of blocks) {
            let w = block.orientation === "vertical" ? block.w :
block.h;
            let h = block.orientation === "vertical" ? block.h :
block.w;
```

```
            let bx = block.x, by = block.y;
            let newAreas = [];
            for (let j = 0; j < areas.length; j++) {
                let area = areas[j];
                if (bx === area.x && by === area.y && w <=
area.w && h <= area.h) {
                    if (area.w - w > 0) {
                        newAreas.push({ x: area.x + w, y:
area.y, w: area.w - w, h: h });
                    }
                    if (area.h - h > 0) {
                        newAreas.push({ x: area.x, y: area.y +
h, w: area.w, h: area.h - h });
                    }
                } else {
                    newAreas.push(area);
                }
            }
            areas = newAreas.filter(a => a.w > 0 && a.h > 0);
        }
        return areas;
    }
    function createRandomGuillotineCutting(blocks, sheetWidth,
sheetHeight) {
        const sheets = [];
        let remaining = blocks.slice();
        const MAX_ATTEMPTS = 1000;
        while (remaining.length > 0) {
            let placed = [];
            let used = Array(remaining.length).fill(false);
            let availableSpaces = [{ x: 0, y: 0, w: sheetWidth,
h: sheetHeight }];
            let attempts = 0;
            for (let i = 0; i < remaining.length; i++) {
                let block = { ...remaining[i] };
                let orientations = ["vertical", "horizontal"];
                let placedBlock = null;
                for (let orientation of orientations) {
                    let w = orientation === "vertical" ? block.w
: block.h;
                    let h = orientation === "vertical" ? block.h
: block.w;
                    for (let j = 0; j < availableSpaces.length;
j++) {
                        let area = availableSpaces[j];
                        if (w <= area.w && h <= area.h) {
                            let x = area.x;
                            let y = area.y;
                            let newBlock = {
                                x,
                                y,
                                orientation,
                                w: block.w,
```

```
                                h: block.h,
                                num: block.num
                            };
                            let ok = true;
                            for (let b of placed) {
                                if
(areRectanglesIntersecting(newBlock, b)) {
                                    ok = false;
                                    break;
                                }
                            }
                            if (!ok) continue;
                            let testPlaced =
placed.concat([newBlock]);
                            if (isGuillotine(testPlaced,
sheetWidth, sheetHeight)) {
                                placedBlock = { ...newBlock };
                                let newSpaces = [];
                                if (area.w - w > 0) {
                                    newSpaces.push({ x: area.x +
w, y: area.y, w: area.w - w, h: h });
                                }
                                if (area.h - h > 0) {
                                    newSpaces.push({ x: area.x,
y: area.y + h, w: area.w, h: area.h - h });
                                }
                                availableSpaces =
availableSpaces.filter((_, idx) => idx !== j).concat(newSpaces);
                                break;
                            }
                        }
                    }
                    if (placedBlock) break;
                }
                if (placedBlock) {
                    placed.push(placedBlock);
                    used[i] = true;
                }
                attempts++;
                if (attempts >= MAX_ATTEMPTS) {
                    break;
                }
            }
            if (placed.length === 0) break;
            sheets.push(placed);
            remaining = remaining.filter((_, idx) =>
!used[idx]);
        }
        return sheets;
    }

    function createInitialPopulation(populationSize, blocks,
sheetWidth, sheetHeight) {
```

```
        const population = [];
        for (let i = 0; i < populationSize; i++) {
            let shuffled = blocks.slice();
            for (let j = shuffled.length - 1; j > 0; j--) {
                let k = Math.floor(Math.random() * (j + 1));
                [shuffled[j], shuffled[k]] = [shuffled[k],
shuffled[j]];
            }
            const randomSheets =
createRandomGuillotineCutting(shuffled, sheetWidth,
sheetHeight);
            population.push({ sheets: randomSheets });
        }
        return population;
    }
    function mutate(individual, mutationRate, sheetWidth,
sheetHeight) {
        let sheets = individual.sheets.map(sheet => sheet.map(b
=> ({ ...b })));
        if (Math.random() < mutationRate) {
            const mutationType = Math.random();
            if (mutationType < 0.25) {
                let sheetIdx = Math.floor(Math.random() *
sheets.length);
                let sheet = sheets[sheetIdx];
                if (sheet.length < 2) return sheets;
                let idx1 = Math.floor(Math.random() *
sheet.length);
                let idx2 = Math.floor(Math.random() *
sheet.length);
                while (idx2 === idx1) idx2 =
Math.floor(Math.random() * sheet.length);
                let newSheet = sheet.slice();
                [newSheet[idx1], newSheet[idx2]] =
[newSheet[idx2], newSheet[idx1]];
                if (isGuillotine(newSheet, sheetWidth,
sheetHeight)) {
                    sheets[sheetIdx] = newSheet;
                }
            }
            else if (mutationType < 0.5) {
                let sheetIdx = Math.floor(Math.random() *
sheets.length);
                let sheet = sheets[sheetIdx];
                if (sheet.length === 0) return sheets;
                let blockIdx = Math.floor(Math.random() *
sheet.length);
                let block = sheet[blockIdx];
                let newOrientation = block.orientation ===
"vertical" ? "horizontal" : "vertical";
                let newBlock = { ...block, orientation:
newOrientation };
                let newSheet = sheet.slice();
```

```
                newSheet[blockIdx] = newBlock;
                if (isGuillotine(newSheet, sheetWidth,
sheetHeight)) {
                    sheets[sheetIdx] = newSheet;
                }
            }
            else if (mutationType < 0.75) {
                let sheetIdx = Math.floor(Math.random() *
sheets.length);
                let sheet = sheets[sheetIdx];
                if (sheet.length === 0) return sheets;
                let blockIdx = Math.floor(Math.random() *
sheet.length);
                let block = sheet[blockIdx];
                let orientations = ["vertical", "horizontal"];
                let otherBlocks = sheet.filter((_, i) => i !==
blockIdx);
                let freeAreas =
getFreeGuillotineAreas(otherBlocks, sheetWidth, sheetHeight);
                let candidates = [];
                for (let orientation of orientations) {
                    let w = orientation === "vertical" ? block.w
: block.h;
                    let h = orientation === "vertical" ? block.h
: block.w;
                    for (let area of freeAreas) {
                        if (w <= area.w && h <= area.h) {
                            candidates.push({
                                x: area.x,
                                y: area.y,
                                orientation
                            });
                        }
                    }
                }
                if (candidates.length > 0) {
                    let chosen =
candidates[Math.floor(Math.random() * candidates.length)];
                    let newBlock = {
                        x: chosen.x,
                        y: chosen.y,
                        orientation: chosen.orientation,
                        w: block.w,
                        h: block.h,
                        num: block.num
                    };
                    let newSheet =
otherBlocks.concat([newBlock]);
                    if (isGuillotine(newSheet, sheetWidth,
sheetHeight)) {
                        sheets[sheetIdx] = newSheet;
                    }
                }
```

```
            }
            else {
                if (sheets.length > 1) {
                    let fromSheetIdx = Math.floor(Math.random()
* sheets.length);
                    let toSheetIdx = Math.floor(Math.random() *
sheets.length);
                    while (toSheetIdx === fromSheetIdx) {
                        toSheetIdx = Math.floor(Math.random() *
sheets.length);
                    }
                    let fromSheet = sheets[fromSheetIdx];
                    let toSheet = sheets[toSheetIdx];
                    if (fromSheet.length > 0) {
                        let blockIdx = Math.floor(Math.random()
* fromSheet.length);
                        let block = fromSheet[blockIdx];
                        let orientations = ["vertical",
"horizontal"];
                        let otherBlocks = toSheet;
                        let freeAreas =
getFreeGuillotineAreas(otherBlocks, sheetWidth, sheetHeight);
                        let candidates = [];
                        for (let orientation of orientations) {
                            let w = orientation === "vertical" ?
block.w : block.h;
                            let h = orientation === "vertical" ?
block.h : block.w;
                            for (let area of freeAreas) {
                                if (w <= area.w && h <= area.h)
{
                                    candidates.push({
                                        x: area.x,
                                        y: area.y,
                                        orientation
                                    });
                                }
                            }
                        }
                        if (candidates.length > 0) {
                            let chosen =
candidates[Math.floor(Math.random() * candidates.length)];
                            let newBlock = {
                                x: chosen.x,
                                y: chosen.y,
                                orientation: chosen.orientation,
                                w: block.w,
                                h: block.h,
                                num: block.num
                            };
                            let newToSheet =
otherBlocks.concat([newBlock]);
                            let newFromSheet =
```

```
fromSheet.slice(0, blockIdx).concat(fromSheet.slice(blockIdx +
1));
                            if (isGuillotine(newToSheet,
sheetWidth, sheetHeight) && isGuillotine(newFromSheet,
sheetWidth, sheetHeight)) {
                                sheets[toSheetIdx] = newToSheet;
                                sheets[fromSheetIdx] =
newFromSheet;
                            }
                        }
                    }
                }
            }
        }
        return sheets;
    }

    // --- Кроссоверu ---

    // One-point crossover
    function onePointCrossover(parent1, parent2, sheetWidth,
sheetHeight) {
        let blocks1 = [];
        parent1.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks1.push({ ...b, sheetIndex: idx })));
        let blocks2 = [];
        parent2.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks2.push({ ...b, sheetIndex: idx })));
        let point = Math.floor(Math.random() * blocks1.length);

        let childBlocks = [];
        for (let i = 0; i < blocks1.length; i++) {
            let src = i < point ? blocks1[i] : blocks2[i];
            let orig = blocks1[i] || src;
            childBlocks.push({
                x: src.x,
                y: src.y,
                orientation: src.orientation,
                w: orig.w,
                h: orig.h,
                num: orig.num,
                sheetIndex: src.sheetIndex
            });
        }
        let maxSheetIndex = Math.max(0, ...childBlocks.map(b =>
b.sheetIndex != null ? b.sheetIndex : 0));
        let sheets = Array.from({ length: maxSheetIndex + 1 },
() => []);
        childBlocks.forEach(b => {
            if (b.sheetIndex != null && sheets[b.sheetIndex]) {
                sheets[b.sheetIndex].push({
                    x: b.x,
                    y: b.y,
```

```
                    orientation: b.orientation,
                    w: b.w,
                    h: b.h,
                    num: b.num
                });
            }
        });
        let valid = true;
        for (let i = 0; i < sheets.length; i++) {
            if (!isGuillotine(sheets[i], sheetWidth,
sheetHeight)) {
                valid = false;
                break;
            }
            for (let j = 0; j < sheets[i].length; j++) {
                let b1 = sheets[i][j];
                if (b1.x < 0 || b1.y < 0 || b1.x +
(b1.orientation === "vertical" ? b1.w : b1.h) > sheetWidth ||
b1.y + (b1.orientation === "vertical" ? b1.h : b1.w) >
sheetHeight) {
                    valid = false;
                    break;
                }
                for (let k = 0; k < sheets[i].length; k++) {
                    if (k === j) continue;
                    let b2 = sheets[i][k];
                    if (areRectanglesIntersecting(b1, b2)) {
                        valid = false;
                        break;
                    }
                }
                if (!valid) break;
            }
            if (!valid) break;
        }
        if (valid) {
            for (let i = 0; i < sheets.length; i++) {
                if (!isGuillotine(sheets[i], sheetWidth,
sheetHeight)) {
                    valid = false;
                    break;
                }
            }
        }
        if (!valid) return [parent1, parent2];
        return [{ sheets }, { sheets: sheets.map(sheet =>
sheet.map(b => ({ ...b }))) }];
    }

    // --- Arithmetic Crossover ---

    function arithmeticCrossover(parent1, parent2, sheetWidth,
sheetHeight) {
```

```
        let alpha = Math.random();
        let blocks1 = [];
        parent1.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks1.push({ ...b, sheetIndex: idx })));
        let blocks2 = [];
        parent2.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks2.push({ ...b, sheetIndex: idx })));
        let childBlocks = [];
        for (let i = 0; i < blocks1.length; i++) {
            let b1 = blocks1[i], b2 = blocks2[i];
            childBlocks.push({
                x: Math.round(alpha * b1.x + (1 - alpha) *
b2.x),
                y: Math.round(alpha * b1.y + (1 - alpha) *
b2.y),
                orientation: Math.random() < 0.5 ?
b1.orientation : b2.orientation,
                w: b1.w,
                h: b1.h,
                num: b1.num,
                sheetIndex: Math.random() < 0.5 ? b1.sheetIndex
: b2.sheetIndex
            });
        }
        let maxSheetIndex = Math.max(0, ...childBlocks.map(b =>
b.sheetIndex != null ? b.sheetIndex : 0));
        let sheets = Array.from({ length: maxSheetIndex + 1 },
() => []);
        childBlocks.forEach(b => {
            if (b.sheetIndex != null && sheets[b.sheetIndex]) {
                sheets[b.sheetIndex].push({
                    x: b.x,
                    y: b.y,
                    orientation: b.orientation,
                    w: b.w,
                    h: b.h,
                    num: b.num
                });
            }
        });
        let valid = true;
        for (let i = 0; i < sheets.length; i++) {
            if (!isGuillotine(sheets[i], sheetWidth,
sheetHeight)) {
                valid = false;
                break;
            }
            for (let j = 0; j < sheets[i].length; j++) {
                let b1 = sheets[i][j];
                if (b1.x < 0 || b1.y < 0 || b1.x +
(b1.orientation === "vertical" ? b1.w : b1.h) > sheetWidth ||
b1.y + (b1.orientation === "vertical" ? b1.h : b1.w) >
sheetHeight) {
```

```
                    valid = false;
                    break;
                }
                for (let k = 0; k < sheets[i].length; k++) {
                    if (k === j) continue;
                    let b2 = sheets[i][k];
                    if (areRectanglesIntersecting(b1, b2)) {
                        valid = false;
                        break;
                    }
                }
                if (!valid) break;
            }
            if (!valid) break;
        }
        if (valid) {
            for (let i = 0; i < sheets.length; i++) {
                if (!isGuillotine(sheets[i], sheetWidth,
sheetHeight)) {
                    valid = false;
                    break;
                }
            }
        }
        if (!valid) return [parent1, parent2];
        return [{ sheets }, { sheets: sheets.map(sheet =>
sheet.map(b => ({ ...b }))) }];
    }


    // --- Order Crossover (OX) ---

    function orderCrossover(parent1, parent2, sheetWidth,
sheetHeight) {
        let blocks1 = [];
        parent1.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks1.push({ ...b, sheetIndex: idx })));
        let blocks2 = [];
        parent2.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks2.push({ ...b, sheetIndex: idx })));
        let size = blocks1.length;
        let start = Math.floor(Math.random() * size);
        let end = Math.floor(Math.random() * size);
        if (start > end) [start, end] = [end, start];

        function ox(parentA, parentB) {
            let child = Array(size).fill(null);
            for (let i = start; i <= end; i++) child[i] =
parentA[i];
            let fillIdx = (end + 1) % size;
            for (let i = 0; i < size; i++) {
                let candidate = parentB[(end + 1 + i) % size];
                if (candidate && !child.some(b => b && b.num ===
```

```
candidate.num)) {
                    child[fillIdx] = candidate;
                    fillIdx = (fillIdx + 1) % size;
                }
            }
            return child.filter(b => b !== null);
        }
        let childBlocks1 = ox(blocks1, blocks2);
        let childBlocks2 = ox(blocks2, blocks1);

        function blocksToSheets(childBlocks) {
            let filteredBlocks = childBlocks.filter(b => b &&
b.sheetIndex != null);
            let maxSheetIndex = Math.max(0,
...filteredBlocks.map(b => b.sheetIndex));
            let sheets = Array.from({ length: maxSheetIndex + 1
}, () => []);
            filteredBlocks.forEach(b => {
                if (sheets[b.sheetIndex]) {
                    sheets[b.sheetIndex].push({
                        x: b.x,
                        y: b.y,
                        orientation: b.orientation,
                        w: b.w,
                        h: b.h,
                        num: b.num
                    });
                }
            });
            return sheets;
        }
        let sheets1 = blocksToSheets(childBlocks1);
        let sheets2 = blocksToSheets(childBlocks2);

        function validSheets(sheets) {
            for (let i = 0; i < sheets.length; i++) {
                if (!isGuillotine(sheets[i], sheetWidth,
sheetHeight)) return false;
                for (let j = 0; j < sheets[i].length; j++) {
                    let b1 = sheets[i][j];
                    if (b1.x < 0 || b1.y < 0 || b1.x +
(b1.orientation === "vertical" ? b1.w : b1.h) > sheetWidth ||
b1.y + (b1.orientation === "vertical" ? b1.h : b1.w) >
sheetHeight) {
                        return false;
                    }
                    for (let k = 0; k < sheets[i].length; k++) {
                        if (k === j) continue;
                        let b2 = sheets[i][k];
                        if (areRectanglesIntersecting(b1, b2))
return false;
                    }
                }
```

```
            }
            return true;
        }
        if (!validSheets(sheets1) || !validSheets(sheets2))
return [parent1, parent2];
        return [{ sheets: sheets1 }, { sheets: sheets2 }];
    }


    // --- Partially Mapped Crossover (PMX) ---
    function pmxCrossover(parent1, parent2, sheetWidth,
sheetHeight) {
        let blocks1 = [];
        parent1.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks1.push({ ...b, sheetIndex: idx })));
        let blocks2 = [];
        parent2.sheets.forEach((sheet, idx) => sheet.forEach(b
=> blocks2.push({ ...b, sheetIndex: idx })));
        let size = blocks1.length;
        let maxAttempts = 10;
        let attempt = 0;

        function pmx(parentA, parentB, start, end) {
            let child = Array(size).fill(null);
            let mapping = {};
            for (let i = start; i <= end; i++) {
                child[i] = parentA[i];
                mapping[parentA[i].num] = parentB[i].num;
            }
            for (let i = 0; i < size; i++) {
                if (i >= start && i <= end) continue;
                let candidate = parentB[i];
                let mappedNum = candidate.num;
                let visited = new Set();
                while
(Object.values(mapping).includes(mappedNum)) {
                    if (visited.has(mappedNum)) break;
                    visited.add(mappedNum);
                    mappedNum = mapping[mappedNum];
                }
                let block = blocks1.find(b => b.num ===
mappedNum) || candidate;
                child[i] = block;
            }
            return child;
        }

        let sheets1, sheets2;
        while (attempt < maxAttempts) {
            let start = Math.floor(Math.random() * size);
            let end = Math.floor(Math.random() * size);
            if (start > end) [start, end] = [end, start];
```

```
            let childBlocks1 = pmx(blocks1, blocks2, start,
end);
            let childBlocks2 = pmx(blocks2, blocks1, start,
end);

            function blocksToSheets(childBlocks) {
                let filteredBlocks = childBlocks.filter(b => b
&& b.sheetIndex != null);
                let maxSheetIndex = Math.max(0,
...filteredBlocks.map(b => b.sheetIndex));
                let sheets = Array.from({ length: maxSheetIndex
+ 1 }, () => []);
                filteredBlocks.forEach(b => {
                    if (sheets[b.sheetIndex]) {
                        sheets[b.sheetIndex].push({
                            x: b.x,
                            y: b.y,
                            orientation: b.orientation,
                            w: b.w,
                            h: b.h,
                            num: b.num
                        });
                    }
                });
                return sheets;
            }
            sheets1 = blocksToSheets(childBlocks1);
            sheets2 = blocksToSheets(childBlocks2);

            function validSheets(sheets) {
                for (let i = 0; i < sheets.length; i++) {
                    if (!isGuillotine(sheets[i], sheetWidth,
sheetHeight)) return false;
                    for (let j = 0; j < sheets[i].length; j++) {
                        let b1 = sheets[i][j];
                        if (b1.x < 0 || b1.y < 0 || b1.x +
(b1.orientation === "vertical" ? b1.w : b1.h) > sheetWidth ||
b1.y + (b1.orientation === "vertical" ? b1.h : b1.w) >
sheetHeight) {
                            return false;
                        }
                        for (let k = 0; k < sheets[i].length;
k++) {
                            if (k === j) continue;
                            let b2 = sheets[i][k];
                            if (areRectanglesIntersecting(b1,
b2)) return false;
                        }
                    }
                }
                return true;
            }
            if (validSheets(sheets1) && validSheets(sheets2)) {
```

```
                return [{ sheets: sheets1 }, { sheets: sheets2
}];
            }
            attempt++;
        }
        return [parent1, parent2];
    }
    function crossover(parent1, parent2, sheetWidth,
sheetHeight) {
        if (crossoverType === 'pmx') {
            return pmxCrossover(parent1, parent2, sheetWidth,
sheetHeight);
        } else if (crossoverType === 'ox') {
            return orderCrossover(parent1, parent2, sheetWidth,
sheetHeight);
        } else if (crossoverType === 'one_point') {
            return onePointCrossover(parent1, parent2,
sheetWidth, sheetHeight);
        } else if (crossoverType === 'arithmetic') {
            return arithmeticCrossover(parent1, parent2,
sheetWidth, sheetHeight);
        } else {
            return pmxCrossover(parent1, parent2, sheetWidth,
sheetHeight);
        }
    }

    function evaluateFitness(sheets, sheetWidth, sheetHeight) {
        let usedArea = 0;
        sheets.forEach(sheet => {
            sheet.forEach(b => {
                usedArea += b.w * b.h;
            });
        });
        let totalArea = sheets.length * sheetWidth *
sheetHeight;
        return (usedArea / totalArea);
    }

    function tournamentSelection(population, tournamentSize) {
        let bestIndividual = population[Math.floor(Math.random()
* population.length)];
        for (let i = 1; i < tournamentSize; i++) {
            const randomIndividual =
population[Math.floor(Math.random() * population.length)];
            if (evaluateFitness(randomIndividual.sheets,
sheetWidth, sheetHeight) >
evaluateFitness(bestIndividual.sheets, sheetWidth, sheetHeight))
{
                bestIndividual = randomIndividual;
            }
        }
        return bestIndividual;
```

```
    }

    function selectParents(population) {
        const parent1 = tournamentSelection(population, 2);
        let parent2 = tournamentSelection(population, 2);
        while (parent2 === parent1) {
            parent2 = tournamentSelection(population, 2);
        }
        return [parent1, parent2];
    }

    let population = initialPopulation &&
Array.isArray(initialPopulation) && initialPopulation.length ===
populationSize
        ? initialPopulation
        : createInitialPopulation(populationSize, blocks,
sheetWidth, sheetHeight);
    let bestFitness = -Infinity;
    let bestIndividual;
    let noImprovementTries = 0;
    const maxNoImprovementTries = 200;

    for (let generation = 0; generation < generations;
generation++) {
        if (noImprovementTries >= maxNoImprovementTries) break;
        const newPopulation = [];
        while (newPopulation.length < populationSize) {
            const [parent1, parent2] =
selectParents(population);
            let [child1, child2] = crossover(parent1, parent2,
sheetWidth, sheetHeight);
            child1.sheets = mutate(child1, mutationRate,
sheetWidth, sheetHeight);
            child2.sheets = mutate(child2, mutationRate,
sheetWidth, sheetHeight);
            newPopulation.push(child1);
            newPopulation.push(child2);
        }
        population = newPopulation;

        let generationBestFitness = -Infinity;
        let generationBestIndividual = null;
        for (let i = 0; i < population.length; i++) {
            const fitness =
evaluateFitness(population[i].sheets, sheetWidth, sheetHeight);
            if (fitness > generationBestFitness) {
                generationBestFitness = fitness;
                generationBestIndividual = population[i];
            }
        }
        if (generationBestFitness > bestFitness) {
            bestFitness = generationBestFitness;
            bestIndividual = generationBestIndividual;
```

```
            noImprovementTries = 0;
        } else {
            noImprovementTries++;
        }
    }

    if (!bestIndividual) {
        for (let i = 0; i < population.length; i++) {
            const fitness =
evaluateFitness(population[i].sheets, sheetWidth, sheetHeight);
            if (fitness > bestFitness) {
                bestFitness = fitness;
                bestIndividual = population[i];
            }
        }
    }

    return bestIndividual.sheets.map(sheet => ({
        width: sheetWidth,
        height: sheetHeight,
        blocks: sheet.map(b => ({
            block: { w: b.w, h: b.h, num: b.num },
            x: b.x,
            y: b.y,
            orientation: b.orientation
        }))
    }));
}
```

Б.1.2 BestFit.js

```
var BestFit = {
  cutBlocks: function (blocks, sheetWidth, sheetHeight) {
    blocks.sort((a, b) => b.area - a.area);
    const sheets = [];
    let currentSheet = new Sheet(sheetWidth, sheetHeight);

    for (const block of blocks) {
      let placed = false;

      for (const sheet of sheets) {
        if (BestFit.placeBlock(block, sheet)) {
          placed = true;
          break;
        }
      }

      if (!placed) {
        if (!BestFit.placeBlock(block, currentSheet)) {
          block.rotate();
          if (!BestFit.placeBlock(block, currentSheet)) {
```

```
            sheets.push(currentSheet);
            currentSheet = new Sheet(sheetWidth, sheetHeight);
            block.rotate();
            BestFit.placeBlock(block, currentSheet);
          }
        }
      }
    }

    sheets.push(currentSheet);
    return sheets;
  },

  placeBlock: function (block, sheet) {
    const fittingPosition = BestFit.findFittingPosition(block,
sheet);
    if (fittingPosition) {
      BestFit.splitSheet(sheet, block, fittingPosition);
      return true;
    }
    return false;
  },

  findFittingPosition: function (block, sheet) {
    if (!sheet.availableSpaces) {
      sheet.availableSpaces = [{ x: 0, y: 0, w: sheet.w, h:
sheet.h }];
    }

    for (const space of sheet.availableSpaces) {
      if (block.w <= space.w && block.h <= space.h) {
        return space;
      }
      if (block.h <= space.w && block.w <= space.h) {
        block.rotate();
        return space;
      }
    }
    return null;
  },

  splitSheet: function (sheet, block, space) {
    block.x = space.x;
    block.y = space.y;
    sheet.blocks.push(block);

    const newSpaces = [];

    if (space.w - block.w > 0) {
      newSpaces.push({ x: space.x + block.w, y: space.y, w:
space.w - block.w, h: space.h });
    }
    if (space.h - block.h > 0) {
```

```
      newSpaces.push({ x: space.x, y: space.y + block.h, w:
block.w, h: space.h - block.h });
    }

    sheet.availableSpaces = sheet.availableSpaces.filter(s => s
!== space).concat(newSpaces);
  }
};
```

Б.1.3 antColonyAlgorithm.js

```
function runACO(blocks, sheetWidth, sheetHeight, antCount = 300,
iterations = 1000, alpha = 1, beta = 2, evaporation = 0.5) {
    const pheromones = {};
    const getKey = (blockIdx, x, y, orientation) =>
`${blockIdx}_${x}_${y}_${orientation}`;
    const step = Math.max(1, Math.floor(Math.min(sheetWidth,
sheetHeight) / 20));
    for (let b = 0; b < blocks.length; b++) {
        for (let orientation of ["vertical", "horizontal"]) {
            let w = orientation === "vertical" ? blocks[b].w :
blocks[b].h;
            let h = orientation === "vertical" ? blocks[b].h :
blocks[b].w;
            for (let x = 0; x <= sheetWidth - w; x += step) {
                for (let y = 0; y <= sheetHeight - h; y += step)
{
                    pheromones[getKey(b, x, y, orientation)] =
1.0;
                }
            }
        }
    }
    let bestSolution = null;
    let bestFitness = -Infinity;
    let noImprovementTries = 0;
    const maxNoImprovementTries = 200;
    for (let iter = 0; iter < iterations; iter++) {
        if (noImprovementTries >= maxNoImprovementTries) break;
        const antSolutions = [];
        for (let ant = 0; ant < antCount; ant++) {
            let remaining = blocks.map((b, idx) => ({ ...b, idx
}));
            let sheets = [];
            while (remaining.length > 0) {
                let placed = [];
                let used = Array(remaining.length).fill(false);
                let availableSpaces = [{ x: 0, y: 0, w:
sheetWidth, h: sheetHeight }];
                for (let i = 0; i < remaining.length; i++) {
                    let block = remaining[i];
```

```
                    let candidates = [];
                    for (let orientation of ["vertical",
"horizontal"]) {
                        let w = orientation === "vertical" ?
block.w : block.h;
                        let h = orientation === "vertical" ?
block.h : block.w;
                        for (let area of availableSpaces) {
                            if (w <= area.w && h <= area.h) {
                                let x = area.x, y = area.y;
                                let heuristic = w * h;
                                let pher =
pheromones[getKey(block.idx, x, y, orientation)] || 1.0;
                                candidates.push({
                                    x, y, orientation, w, h,
idx: block.idx,
                                    score: Math.pow(pher, alpha)
* Math.pow(heuristic, beta)
                                });
                            }
                        }
                    }
                    if (candidates.length === 0) continue;
                    let sumScore = candidates.reduce((s, c) => s
+ c.score, 0);
                    let r = Math.random() * sumScore;
                    let acc = 0, chosen = null;
                    for (let c of candidates) {
                        acc += c.score;
                        if (acc >= r) {
                            chosen = c;
                            break;
                        }
                    }
                    if (!chosen) chosen = candidates[0];
                    let ok = true;
                    for (let b of placed) {
                        if (areRectanglesIntersecting(
                            { x: chosen.x, y: chosen.y, w:
block.w, h: block.h, orientation: chosen.orientation },
                            { x: b.x, y: b.y, w: b.w, h: b.h,
orientation: b.orientation }
                        )) {
                            ok = false;
                            break;
                        }
                    }
                    if (!ok) continue;
                    let testPlaced = placed.concat([{
                        x: chosen.x, y: chosen.y, w: block.w, h:
block.h, orientation: chosen.orientation, num: block.num
                    }]);
                    if (!isGuillotine(testPlaced, sheetWidth,
```

```
sheetHeight)) continue;
                    placed.push({
                        x: chosen.x, y: chosen.y, w: block.w, h:
block.h, orientation: chosen.orientation, num: block.num
                    });
                    used[i] = true;
                    let areaIdx = availableSpaces.findIndex(a =>
a.x === chosen.x && a.y === chosen.y && a.w >= chosen.w && a.h
>= chosen.h);
                    if (areaIdx !== -1) {
                        let area = availableSpaces[areaIdx];
                        let newSpaces = [];
                        if (area.w - chosen.w > 0) {
                            newSpaces.push({ x: area.x +
chosen.w, y: area.y, w: area.w - chosen.w, h: chosen.h });
                        }
                        if (area.h - chosen.h > 0) {
                            newSpaces.push({ x: area.x, y:
area.y + chosen.h, w: area.w, h: area.h - chosen.h });
                        }
                        availableSpaces =
availableSpaces.filter((_, idx) => idx !==
areaIdx).concat(newSpaces);
                    }
                }
                if (placed.length === 0) break;
                sheets.push(placed);
                remaining = remaining.filter((_, idx) =>
!used[idx]);
            }
            antSolutions.push(sheets);
        }
        for (let key in pheromones) {
            pheromones[key] *= (1 - evaporation);
            if (pheromones[key] < 1e-6) pheromones[key] = 1e-6;
        }
        let improved = false;
        for (let sheets of antSolutions) {
            let fitness = calculateFitness(sheets, sheetWidth,
sheetHeight);
            if (fitness > bestFitness) {
                bestFitness = fitness;
                bestSolution = sheets;
                improved = true;
            }
            sheets.forEach((sheet) => {
                sheet.forEach((b, idx) => {
                    let key = getKey(idx, b.x, b.y,
b.orientation);
                    pheromones[key] += fitness;
                });
            });
        }
```

```
        if (improved) {
            noImprovementTries = 0;
        } else {
            noImprovementTries++;
        }
    }
    return bestSolution.map(sheet => ({
        width: sheetWidth,
        height: sheetHeight,
        blocks: sheet.map(b => ({
            block: { w: b.w, h: b.h, num: b.num },
            x: b.x,
            y: b.y,
            orientation: b.orientation
        }))
    }));
    function areRectanglesIntersecting(rect1, rect2) {
        let rect1Width = rect1.orientation === "vertical" ?
rect1.w : rect1.h;
        let rect1Height = rect1.orientation === "vertical" ?
rect1.h : rect1.w;
        let rect2Width = rect2.orientation === "vertical" ?
rect2.w : rect2.h;
        let rect2Height = rect2.orientation === "vertical" ?
rect2.h : rect2.w;
        if (
            rect1.x + rect1Width <= rect2.x ||
            rect2.x + rect2Width <= rect1.x ||
            rect1.y + rect1Height <= rect2.y ||
            rect2.y + rect2Height <= rect1.y
        ) {
            return false;
        }
        return true;
    }

    function isGuillotine(blocks, sheetWidth, sheetHeight) {
        let areas = [{ x: 0, y: 0, w: sheetWidth, h: sheetHeight
}];
        for (let i = 0; i < blocks.length; i++) {
            let block = blocks[i];
            let w = block.orientation === "vertical" ? block.w :
block.h;
            let h = block.orientation === "vertical" ? block.h :
block.w;
            let bx = block.x, by = block.y;
            let found = false;
            for (let j = 0; j < areas.length; j++) {
                let area = areas[j];
                if (
                    bx === area.x && by === area.y &&
                    w <= area.w && h <= area.h
                ) {
```

```
                    let ok = true;
                    for (let k = 0; k < i; k++) {
                        let prev = blocks[k];
                        if (areRectanglesIntersecting(block,
prev)) {
                            ok = false;
                            break;
                        }
                    }
                    if (!ok) continue;

                    let newAreas = [];
                    if (area.w - w > 0) {
                        newAreas.push({ x: area.x + w, y:
area.y, w: area.w - w, h: h });
                    }
                    if (area.h - h > 0) {
                        newAreas.push({ x: area.x, y: area.y +
h, w: area.w, h: area.h - h });
                    }
                    for (let k = 0; k < areas.length; k++) {
                        if (k !== j) newAreas.push(areas[k]);
                    }
                    areas = newAreas.filter(a => a.w > 0 && a.h
> 0);
                    found = true;
                    break;
                }
            }
            if (!found) return false;
        }
        return true;
    }

    function calculateFitness(sheets, sheetWidth, sheetHeight) {
        let usedArea = 0;
        sheets.forEach(sheet => {
            sheet.forEach(b => {
                usedArea += b.w * b.h;
            });
        });
        let totalArea = sheets.length * sheetWidth *
sheetHeight;
        return usedArea / totalArea;
    }
}
```

Б.1.4 simulatedAnnealing.js

```
function areRectanglesIntersecting(rect1, rect2) {
    let rect1Width = rect1.orientation === "vertical" ? rect1.w
```

```
: rect1.h;
    let rect1Height = rect1.orientation === "vertical" ? rect1.h
: rect1.w;
    let rect2Width = rect2.orientation === "vertical" ? rect2.w
: rect2.h;
    let rect2Height = rect2.orientation === "vertical" ? rect2.h
: rect2.w;
    if (
        rect1.x + rect1Width <= rect2.x ||
        rect2.x + rect2Width <= rect1.x ||
        rect1.y + rect1Height <= rect2.y ||
        rect2.y + rect2Height <= rect1.y
    ) {
        return false;
    }
    return true;
}

function isGuillotine(blocks, sheetWidth, sheetHeight) {
    let areas = [{ x: 0, y: 0, w: sheetWidth, h: sheetHeight }];
    for (let i = 0; i < blocks.length; i++) {
        let block = blocks[i];
        let w = block.orientation === "vertical" ? block.w :
block.h;
        let h = block.orientation === "vertical" ? block.h :
block.w;
        let bx = block.x, by = block.y;
        let found = false;
        for (let j = 0; j < areas.length; j++) {
            let area = areas[j];
            if (
                bx === area.x && by === area.y &&
                w <= area.w && h <= area.h
            ) {
                let ok = true;
                for (let k = 0; k < i; k++) {
                    let prev = blocks[k];
                    if (areRectanglesIntersecting(block, prev))
{
                        ok = false;
                        break;
                    }
                }
                if (!ok) continue;

                let newAreas = [];
                if (area.w - w > 0) {
                    newAreas.push({ x: area.x + w, y: area.y, w:
area.w - w, h: h });
                }
                if (area.h - h > 0) {
                    newAreas.push({ x: area.x, y: area.y + h, w:
area.w, h: area.h - h });
```

```
                }
                for (let k = 0; k < areas.length; k++) {
                    if (k !== j) newAreas.push(areas[k]);
                }
                areas = newAreas.filter(a => a.w > 0 && a.h >
0);
                found = true;
                break;
            }
        }
        if (!found) return false;
    }
    return true;
}

function getFreeGuillotineAreas(blocks, sheetWidth, sheetHeight)
{
    let areas = [{ x: 0, y: 0, w: sheetWidth, h: sheetHeight }];
    for (const block of blocks) {
        let w = block.orientation === "vertical" ? block.w :
block.h;
        let h = block.orientation === "vertical" ? block.h :
block.w;
        let bx = block.x, by = block.y;
        let newAreas = [];
        for (let j = 0; j < areas.length; j++) {
            let area = areas[j];
            if (bx === area.x && by === area.y && w <= area.w &&
h <= area.h) {
                if (area.w - w > 0) {
                    newAreas.push({ x: area.x + w, y: area.y, w:
area.w - w, h: h });
                }
                if (area.h - h > 0) {
                    newAreas.push({ x: area.x, y: area.y + h, w:
area.w, h: area.h - h });
                }
            } else {
                newAreas.push(area);
            }
        }
        areas = newAreas.filter(a => a.w > 0 && a.h > 0);
    }
    return areas;
}

function generateInitialSolution(blocks, sheetWidth,
sheetHeight) {
    let remaining = blocks.slice();
    let sheets = [];
    while (remaining.length > 0) {
        let placed = [];
        let used = Array(remaining.length).fill(false);
```

```
        let availableSpaces = [{ x: 0, y: 0, w: sheetWidth, h:
sheetHeight }];
        for (let i = 0; i < remaining.length; i++) {
            let block = { ...remaining[i] };
            let orientations = ["vertical", "horizontal"];
            let placedBlock = null;
            for (let orientation of orientations) {
                let w = orientation === "vertical" ? block.w :
block.h;
                let h = orientation === "vertical" ? block.h :
block.w;
                for (let j = 0; j < availableSpaces.length; j++)
{
                    let area = availableSpaces[j];
                    if (w <= area.w && h <= area.h) {
                        let x = area.x, y = area.y;
                        let newBlock = {
                            x, y, orientation,
                            w: block.w, h: block.h, num:
block.num
                        };
                        let ok = true;
                        for (let b of placed) {
                            if
(areRectanglesIntersecting(newBlock, b)) {
                                ok = false;
                                break;
                            }
                        }
                        if (!ok) continue;
                        let testPlaced =
placed.concat([newBlock]);
                        if (isGuillotine(testPlaced, sheetWidth,
sheetHeight)) {
                            placedBlock = { ...newBlock };
                            let newSpaces = [];
                            if (area.w - w > 0) {
                                newSpaces.push({ x: area.x + w,
y: area.y, w: area.w - w, h: h });
                            }
                            if (area.h - h > 0) {
                                newSpaces.push({ x: area.x, y:
area.y + h, w: area.w, h: area.h - h });
                            }
                            availableSpaces =
availableSpaces.filter((_, idx) => idx !== j).concat(newSpaces);
                            break;
                        }
                    }
                }
                if (placedBlock) break;
            }
            if (placedBlock) {
```

```
                placed.push(placedBlock);
                used[i] = true;
            }
        }
        if (placed.length === 0) break;
        sheets.push(placed);
        remaining = remaining.filter((_, idx) => !used[idx]);
    }
    return sheets;
}
function generateNeighbor(sheets, sheetWidth, sheetHeight) {
    let newSheets = sheets.map(sheet => sheet.map(b => ({ ...b
}));
    let mutationType = Math.random();
    if (mutationType < 0.33) {
        let sheetIdx = Math.floor(Math.random() *
newSheets.length);
        let sheet = newSheets[sheetIdx];
        if (sheet.length < 2) return newSheets;
        let idx1 = Math.floor(Math.random() * sheet.length);
        let idx2 = Math.floor(Math.random() * sheet.length);
        while (idx2 === idx1) idx2 = Math.floor(Math.random() *
sheet.length);
        let newSheet = sheet.slice();
        [newSheet[idx1], newSheet[idx2]] = [newSheet[idx2],
newSheet[idx1]];
        if (isGuillotine(newSheet, sheetWidth, sheetHeight)) {
            newSheets[sheetIdx] = newSheet;
        }
    }
    else if (mutationType < 0.66) {
        let sheetIdx = Math.floor(Math.random() *
newSheets.length);
        let sheet = newSheets[sheetIdx];
        if (sheet.length === 0) return newSheets;
        let blockIdx = Math.floor(Math.random() * sheet.length);
        let block = sheet[blockIdx];
        let newOrientation = block.orientation === "vertical" ?
"horizontal" : "vertical";
        let newBlock = { ...block, orientation: newOrientation
};
        let newSheet = sheet.slice();
        newSheet[blockIdx] = newBlock;
        if (isGuillotine(newSheet, sheetWidth, sheetHeight)) {
            newSheets[sheetIdx] = newSheet;
        }
    }
    else {
        let sheetIdx = Math.floor(Math.random() *
newSheets.length);
        let sheet = newSheets[sheetIdx];
        if (sheet.length === 0) return newSheets;
        let blockIdx = Math.floor(Math.random() * sheet.length);
```

```
        let block = sheet[blockIdx];
        let orientations = ["vertical", "horizontal"];
        let otherBlocks = sheet.filter((_, i) => i !==
blockIdx);
        let freeAreas = getFreeGuillotineAreas(otherBlocks,
sheetWidth, sheetHeight);
        let candidates = [];
        for (let orientation of orientations) {
            let w = orientation === "vertical" ? block.w :
block.h;
            let h = orientation === "vertical" ? block.h :
block.w;
            for (let area of freeAreas) {
                if (w <= area.w && h <= area.h) {
                    candidates.push({
                        x: area.x,
                        y: area.y,
                        orientation
                    });
                }
            }
        }
        if (candidates.length > 0) {
            let chosen = candidates[Math.floor(Math.random() *
candidates.length)];
            let newBlock = {
                x: chosen.x,
                y: chosen.y,
                orientation: chosen.orientation,
                w: block.w,
                h: block.h,
                num: block.num
            };
            let newSheet = otherBlocks.concat([newBlock]);
            if (isGuillotine(newSheet, sheetWidth, sheetHeight))
{
                newSheets[sheetIdx] = newSheet;
            }
        }
    }
    return newSheets;
}
function calculateEnergy(sheets, sheetWidth, sheetHeight) {
    let usedArea = 0;
    sheets.forEach(sheet => {
        sheet.forEach(b => {
            usedArea += b.w * b.h;
        });
    });
    let totalArea = sheets.length * sheetWidth * sheetHeight;
    return usedArea / totalArea;
}
function runSimulatedAnnealing(blocks, sheetWidth, sheetHeight,
```

```
options = {})
{
    const {
        initialTemperature = 1000.0,
        finalTemperature = 1e-4,
        alpha = 0.98,
        maxIterations = 1000
    } = options;

    let currentSolution = generateInitialSolution(blocks,
sheetWidth, sheetHeight);
    let currentEnergy = calculateEnergy(currentSolution,
sheetWidth, sheetHeight);
    let bestSolution = currentSolution;
    let bestEnergy = currentEnergy;
    let temperature = initialTemperature;

    for (let iter = 0; iter < maxIterations && temperature >
finalTemperature; iter++)
{
        let newSolution = generateNeighbor(currentSolution,
sheetWidth, sheetHeight);
        let newEnergy = calculateEnergy(newSolution, sheetWidth,
sheetHeight);
        if (newEnergy > currentEnergy || Math.random() <
Math.exp((newEnergy - currentEnergy) / temperature)) {
            currentSolution = newSolution;
            currentEnergy = newEnergy;
            if (newEnergy > bestEnergy) {
                bestSolution = newSolution;
                bestEnergy = newEnergy;
            }
        }
        temperature *= alpha;
    }
    return bestSolution.map(sheet => ({
        width: sheetWidth,
        height: sheetHeight,
        blocks: sheet.map(b => ({
            block: { w: b.w, h: b.h, num: b.num },
            x: b.x,
            y: b.y,
            orientation: b.orientation
        }))
    }));
}
```

Б.1.5 hybridGeneticAlgorithm.js

```
function runHybridGeneticAlgorithm(populationSize, mutationRate,
```

```
generations, blocks, sheetWidth, sheetHeight, crossoverType) {
    function createBestFitPopulation(populationSize, blocks,
sheetWidth, sheetHeight) {
        const population = [];
        for (let i = 0; i < populationSize; i++) {
            let shuffled = blocks.slice();
            for (let j = shuffled.length - 1; j > 0; j--) {
                let k = Math.floor(Math.random() * (j + 1));
                [shuffled[j], shuffled[k]] = [shuffled[k],
shuffled[j]];
            }
            let sheets = BestFit.cutBlocks(shuffled, sheetWidth,
sheetHeight).map(sheet => {
                return sheet.blocks.map(b => ({
                    x: b.x,
                    y: b.y,
                    orientation: "vertical",
                    w: b.w,
                    h: b.h,
                    num: b.num
                }));
            });
            population.push({ sheets });
        }
        return population;
    }
    const initialPopulation =
createBestFitPopulation(populationSize, blocks, sheetWidth,
sheetHeight);
    return runGeneticAlgorithm(populationSize, mutationRate,
generations, blocks, sheetWidth, sheetHeight, initialPopulation,
crossoverType);
}
```

Б.2 Вихідний код класів

Б.2.1 Класс Block

```
class Block {
  constructor(w, h, num) {
    this.w = w; // Width of the block
    this.h = h; // Height of the block
    //this.isPlaced = false; // Indicates if the block is placed
on a sheet
    this.x = 0; // X-coordinate of the block's position on the
sheet
    this.y = 0; // Y-coordinate of the block's position on the
sheet
    this.num = num; // A unique identifier for the block
```

```
    this.area = w * h; // Area of the block
    this.rotation = 0; // 0 - horizontal, 1 - vertical (current
rotation state)
  }

  // Method to rotate the block
  rotate() {
    [this.w, this.h] = [this.h, this.w]; // Swap the width and
height values
    this.rotation = this.rotation === 0 ? 1 : 0; // Toggle the
rotation state between horizontal and vertical
  }

}
```

Б.2.2 Класс Sheet

```
class Sheet {
  constructor(w, h) {
    this.w = w; // Width of the sheet
    this.h = h; // Height of the sheet
    this.blocks = []; // Array to store placed blocks on the
sheet
  }
```

Б.3 Додаткові модулі

Б.3.1 Модуль details.js

```
var details = {
    list: [],

    add_block: function() {
      var result = [];
      if((parseInt(Demo.el.block_w.val()) <=
parseInt(Demo.el.size_w.val()) &&
parseInt(Demo.el.block_h.val()) <= (Demo.el.size_h.val())) ||
(parseInt(Demo.el.block_h.val()) <=
parseInt(Demo.el.size_w.val()) &&
parseInt(Demo.el.block_w.val()) <= (Demo.el.size_h.val()))){
        result.push( new Block(parseInt(Demo.el.block_w.val()),
parseInt(Demo.el.block_h.val()),
parseInt(Demo.el.block_n.val()),));
      }

      for (i = 0 ; i < result.length ; i++) {
        for (j = 0 ; j < result[i].num ; j++) {
          var newNum;
          if (details.list.length == 0) {
```

```
            newNum = 1;
          } else {
            newNum =  details.list[details.list.length - 1].num
+ 1;
          }
          details.list.push(new Block(result[i].w, result[i].h,
newNum));
        }
      }
      Demo.run();
    },

    edit_block: function(){
      var result = [];
      let exist = false;
      let newEl = new
Block(parseInt(Demo.el.edit_block_w.val()),
parseInt(Demo.el.edit_block_h.val()),
parseInt(Demo.el.edit_block_n.val()));

      for(const item of details.list)
      {
        if(item.num == newEl.num)
        {
          item.w = newEl.w;
          item.h = newEl.h;
          exist = true;
        }
      }

      if(!exist && !(newEl.num > details.list.length)){
        details.list.splice(newEl.num-1, 0, newEl)
      }

      Demo.run();
    },

    del_block: function(id) {
      var newArray = details.list.filter(el => el.num != id);
      details.list.length = 0;
      newArray.forEach(element => {
        details.list.push(element);
      });
      Demo.run();
    },

    print_det: function() {
      var det = document.getElementById("details");
      det.innerHTML = '';
      details.list.forEach(element => {
        var closeButton = document.createElement('button');
        closeButton.className = 'delButton';
        closeButton.id = element.num;
```

```
        closeButton.innerText = 'X';
        closeButton.addEventListener('click', function() {
          details.del_block(closeButton.id);
        });

        var textNode = document.createTextNode(element.num + ".
" + element.w + " x " + element.h);

        var container = document.createElement('div');
        container.append(textNode);
        container.append(closeButton);


        Demo.el.details.append(container);
      });
    },

    populateDropdownList: function(element, values) {
      var dropdownList =
document.getElementById(element).nextElementSibling;

      dropdownList.innerHTML = '';

      values.forEach(function(value) {
        var listItem = document.createElement('li');
        listItem.textContent = value;
        dropdownList.appendChild(listItem);
      });

      if(dropdownList.style.display === "block"){
        dropdownList.style.display = "none";
      }else{
        dropdownList.style.display = "block";
      }

      dropdownList.addEventListener('click', function(event) {
        if (event.target.tagName === 'LI') {
          var selectedValue = event.target.textContent;
          document.getElementById(element).value =
selectedValue;
          dropdownList.style.display = "none";
          Demo.run();
        }
      });
    },

    saveValuesToLocalStorage: function(name) {

      var maxValues = 4;

      switch (name) {
        case 'block_h':
          var block_h = Demo.el.block_h.val();
```

```
    var savedH = localStorage.getItem('block_h') || '';

    var valuesH = savedH.split(',');
    valuesH.unshift(block_h);

    if (valuesH.length > maxValues) {
      valuesH = valuesH.slice(0, maxValues);
    }
    localStorage.setItem('block_h', valuesH.join(','));

    break;
  case 'block_w':
    var block_w = Demo.el.block_w.val();

    var savedW = localStorage.getItem('block_w') || '';

    var valuesW = savedW.split(',');
    valuesW.unshift(block_w);

    if (valuesW.length > maxValues) {
      valuesW = valuesW.slice(0, maxValues);
    }

    localStorage.setItem('block_w', valuesW.join(','));

    break;
  case 'block_n':
    var block_n = Demo.el.block_n.val();

    var savedN = localStorage.getItem('block_n') || '';

    var valuesN = savedN.split(',');
    valuesN.unshift(block_n);
    if (valuesN.length > maxValues) {
      valuesN = valuesN.slice(0, maxValues);
    }

    localStorage.setItem('block_n', valuesN.join(','));

    break;
  case 'size_h':
    var size_h = Demo.el.size_h.val();

    var savedSH = localStorage.getItem('size_h') || '';

    var valuesSH = savedSH.split(',');
    valuesSH.unshift(size_h);

    if (valuesSH.length > maxValues) {
      valuesSH = valuesSH.slice(0, maxValues);
    }
    localStorage.setItem('size_h', valuesSH.join(','));
```

```
          break;
        case 'size_w':
          var size_w = Demo.el.size_w.val();

          var savedSW = localStorage.getItem('size_w') || '';
          var valuesSW = savedSW.split(',');
          valuesSW.unshift(size_w);

          if (valuesSW.length > maxValues) {
            valuesSW = valuesSW.slice(0, maxValues);
          }

          localStorage.setItem('size_w', valuesSW.join(','));

          break;
      }
    },

    report: function(sheets, blocks, w, h ) {
      var fit = 0, block, n, len = blocks.length;
      for (n = 0 ; n < len ; n++) {
         block = blocks[n];
          fit += block.area;
      }
      Demo.el.ratio.text(Math.round(100 * fit / (w * h)));
      Demo.el.numSheets.html(("Кількість листів (шт): " +
sheets));
    },
  };
```

Б.3.2 Модуль canvas.js

```
var canvas = {
    reset: function(width, height) {
      Demo.el.canvas.width  = width  + 1;
      Demo.el.canvas.height = height + 1;
      Demo.el.draw.clearRect(0, 0, Demo.el.canvas.width,
Demo.el.canvas.height);
    },

    rect:  function(x, y, w, h, color, num) {
      Demo.el.draw.fillStyle = color;
      Demo.el.draw.fillRect(x + 0.5, y + 0.5, w, h);
      Demo.el.draw.fillStyle = '#222';
      Demo.el.draw.font = "13px Inter, Arial, sans-serif";
      Demo.el.draw.textAlign = "center";
      Demo.el.draw.textBaseline = "middle";
      Demo.el.draw.fillText(
        num.toString(),
        x + 0.5 + w / 2,
        y + 0.5 + h / 2
```

```
      );
    },

    stroke: function(x, y, w, h) {
      Demo.el.draw.save();
      Demo.el.draw.strokeStyle = "#90caf9";
      Demo.el.draw.lineWidth = 1;
      Demo.el.draw.strokeRect(x + 0.5, y + 0.5, w, h);
      Demo.el.draw.restore();
    },

    blocks: function(sheets, algorithm) {
        var diffSheets = 0;
        for (let i = 0; i < sheets.length; i++) {
          for (let n = 0 ; n < sheets[i].blocks.length ; n++) {
            var block = sheets[i].blocks[n];
            if (algorithm === "genetic" || algorithm ===
"hybrid_genetic" || algorithm === "simulated_annealing" ||
algorithm === "ant_colony") {
              if (block.orientation === "vertical") {
                  canvas.rect(block.x, block.y + diffSheets,
block.block.w, block.block.h, canvas.color(n), block.block.num);
                  canvas.stroke(block.x, block.y + diffSheets,
block.block.w, block.block.h);
              }else{
                  canvas.rect(block.x, block.y + diffSheets,
block.block.h, block.block.w, canvas.color(n), block.block.num);
                  canvas.stroke(block.x, block.y + diffSheets,
block.block.h, block.block.w);
                }
            } else {
              canvas.rect(block.x, block.y + diffSheets,
block.w, block.h, canvas.color(n), block.num);
              canvas.stroke(block.x, block.y + diffSheets,
block.w, block.h);
            }
          }
          Demo.el.draw.save();
          Demo.el.draw.strokeStyle = "#cccccc";
          Demo.el.draw.lineWidth = 1;
          Demo.el.draw.strokeRect(0, diffSheets,
parseInt(Demo.el.size_w.val()), 1);
          Demo.el.draw.restore();
          diffSheets += parseInt(Demo.el.size_h.val());
        }
        canvas.calculateEfficiency(sheets);
    },

    calculateEfficiency: function(sheets) {
        let minX = Infinity, minY = Infinity, maxX = 0, maxY =
0;
        sheets.forEach(sheet => {
            sheet.blocks.forEach(block => {
```

```
                minX = Math.min(minX, block.x);
                minY = Math.min(minY, block.y);
                maxX = Math.max(maxX, block.x + (block.w ?
block.w : block.block.w));
                maxY = Math.max(maxY, block.y + (block.h ?
block.h : block.block.h));
            });
        });
        let totalArea = (maxX - minX) * (maxY - minY);
        let usedArea = sheets.reduce((sum, sheet) => {
            return sum + sheet.blocks.reduce((blockSum, block)
=> {
                return blockSum + (block.w ? block.w * block.h :
block.block.w * block.block.h);
            }, 0);
        }, 0);
        let efficiency = (usedArea / totalArea) * 100 /
sheets.length;
        Demo.el.efficiency.text(`Оцінка:
${efficiency.toFixed(2)}%`);
    },

    saveToPdf: function () {
      var doc = new jsPDF();

      var imgData = Demo.el.canvas.toDataURL('image/png');

        var canvasWidth = Demo.el.canvas.width;
        var canvasHeight = Demo.el.canvas.height;

        var availableWidth = 210;
        var availableHeight = 290;

        var scale = Math.min(availableWidth / canvasWidth,
availableHeight / canvasHeight);
        var imageWidth = canvasWidth * scale;
        var imageHeight = canvasHeight * scale;

      doc.addImage(imgData, 'PNG', 0, 0, imageWidth,
imageHeight);

      doc.save('canvas_image.pdf');
    },

    color: function(n) {
        var cols =["#b3e5fc"];
        return cols[n % cols.length];
      },
  };
```

Б.3.3 Модуль demo.js

```
import { runGeneticAlgorithm, } from
'../Model/geneticAlgorithm.js';
import { Block } from '../Model/Block.js';
import  { runACO } from '../Model/antColonyAlgorithm.js'; //
Update import for CommonJS module
import { runSimulatedAnnealing } from
'../Model/simulatedAnnealing.js';
import { BestFit } from '../Model/BestFit.js';
import { runHybridGeneticAlgorithm } from
'../Model/hybridGeneticAlgorithm.js';

var Demo = {

  init: function () {
    // Initialize DOM elements
    Demo.el = {
      details: $('#details'),
      block_h: $('#block_h'),
      block_w: $('#block_w'),
      block_n: $('#block_n'),
      canvas: $('#canvas')[0],
      size_h: $('#size_h'),
      size_w: $('#size_w'),
      numSheets: $('#numSheets'),
      ratio: $('#ratio'),
      edit_block_n: $('#edit_block_n'),
      edit_block_w: $('#edit_block_w'),
      edit_block_h: $('#edit_block_h'),
      add_block: $('#add_block'),
      edit_block: $('#edit_block'),
      run_button: $('#run_button'),
      algorithm_select: $('#algorithm_select'),
      execution_time: $('#execution_time'),
      efficiency: $('#efficiency'),
      toggle_details: $('#toggle_details')

    if (!Demo.el.canvas.getContext)
      return false;
    Demo.el.draw = Demo.el.canvas.getContext("2d");

    $(Demo.el.add_block).click(function (event) {
      if (parseInt(Demo.el.size_h.val()) <= 3000 &&
parseInt(Demo.el.size_w.val()) <= 3000) {
        details.add_block();
        Demo.run();
      } else {
        var el = document.getElementById('size_report');
        el.style.display = 'block';
      }
    });
```

```
    $(Demo.el.edit_block).click(function (event) {
      details.edit_block();
      Demo.run();
    });
    Demo.el.size_h.change(function (ev) {
      if (parseInt(Demo.el.size_h.val()) <= 3000) {
        details.saveValuesToLocalStorage('size_h');
        var el = document.getElementById('size_report');
        el.style.display = 'none';
      } else {
        var el = document.getElementById('size_report');
        el.style.display = 'block';
      }
    });

    Demo.el.size_w.change(function (ev) {
      if (parseInt(Demo.el.size_h.val()) <= 3000) {
        details.saveValuesToLocalStorage('size_w');
        var el = document.getElementById('size_report');
        el.style.display = 'none';
      } else {
        var el = document.getElementById('size_report');
        el.style.display = 'block';
      }
    });

    Demo.el.block_h.change(function (ev) {
      details.saveValuesToLocalStorage('block_h');
    });

    Demo.el.block_w.change(function (ev) {
      details.saveValuesToLocalStorage('block_w');
    });

    Demo.el.block_n.change(function (ev) {
      details.saveValuesToLocalStorage('block_n');
    });
    $(Demo.el.size_h).click(function (event) {
      var savedSH = localStorage.getItem('size_h');
      details.populateDropdownList('size_h', savedSH ?
savedSH.split(',') : []);
    });

    $(Demo.el.size_w).click(function (event) {
      var savedSW = localStorage.getItem('size_w');
      details.populateDropdownList('size_w', savedSW ?
savedSW.split(',') : []);
    });

    $(Demo.el.block_w).click(function (event) {
      var savedW = localStorage.getItem('block_w');
      details.populateDropdownList('block_w', savedW ?
savedW.split(',') : []);
```

```
    });

    $(Demo.el.block_h).click(function (event) {
      var savedH = localStorage.getItem('block_h');
      details.populateDropdownList('block_h', savedH ?
savedH.split(',') : []);
    });

    $(Demo.el.block_n).click(function (event) {
      var savedN = localStorage.getItem('block_n');
      details.populateDropdownList('block_n', savedN ?
savedN.split(',') : []);
    });
    $(Demo.el.run_button).click(function (event) {
      Demo.run();
    });

    $(Demo.el.toggle_details).change(function (event) {
      Demo.el.details.toggleClass('hidden');
    });
  },

  run: function () {
    var newBlocks = details.list.slice(0);
    if (newBlocks.length > 0) {
      var algorithm = Demo.el.algorithm_select.val();
      var resSheets;
      var startTime = performance.now();
      if (algorithm === "genetic") {
        resSheets = runGeneticAlgorithm(5000, 0.01, 5000,
newBlocks, parseInt(Demo.el.size_w.val()),
parseInt(Demo.el.size_h.val()), undefined, 'ox');
      } else if (algorithm === "best_fit") {
        resSheets = BestFit.cutBlocks(newBlocks,
parseInt(Demo.el.size_w.val()), parseInt(Demo.el.size_h.val()));
      } else if (algorithm === "ant_colony") {
        resSheets = runACO(newBlocks,
parseInt(Demo.el.size_w.val()), parseInt(Demo.el.size_h.val()),
500, 1000, 1, 2, 0.5);
      } else if (algorithm === "simulated_annealing") {
        resSheets = runSimulatedAnnealing(newBlocks,
parseInt(Demo.el.size_w.val()), parseInt(Demo.el.size_h.val()));
      } else if (algorithm === "hybrid_genetic") {
        resSheets = runHybridGeneticAlgorithm(5000, 0.01, 5000,
newBlocks, parseInt(Demo.el.size_w.val()),
parseInt(Demo.el.size_h.val()));
      }
      var endTime = performance.now();
      var executionTime = endTime - startTime;
      Demo.el.execution_time.text(`Execution Time:
${executionTime.toFixed(2)} ms`);

      var hCanvas = parseInt(Demo.el.size_h.val()) *
```

```
resSheets.length;
      canvas.reset(parseInt(Demo.el.size_w.val()), hCanvas);
      canvas.blocks(resSheets, algorithm);
      details.report(resSheets.length, newBlocks,
parseInt(Demo.el.size_w.val()), hCanvas);
      details.print_det();
    }
  },
}

function getTestBlocks(n) {
  const base = [
    new Block(50, 50, 1), new Block(50, 50, 2), new Block(60,
40, 3), new Block(60, 40, 4),
    new Block(70, 30, 5), new Block(10, 40, 6), new Block(10,
40, 7), new Block(30, 50, 8),
    new Block(30, 50, 9), new Block(80, 10, 10), new Block(10,
20, 11), new Block(10, 20, 12),
    new Block(10, 20, 13), new Block(10, 20, 14), new Block(30,
20, 15), new Block(30, 20, 16),
    new Block(75, 13, 17), new Block(75, 13, 18), new Block(12,
46, 19), new Block(12, 46, 20),
    new Block(45, 27, 21), new Block(45, 27, 22), new Block(56,
77, 23), new Block(56, 77, 24),
    new Block(83, 55, 25), new Block(83, 55, 26), new Block(23,
21, 27), new Block(23, 21, 28),
    new Block(33, 33, 29), new Block(33, 33, 30), new Block(63,
33, 31), new Block(63, 33, 32),
    new Block(19, 24, 33), new Block(19, 24, 34), new Block(24,
10, 35), new Block(24, 10, 36),
    new Block(56, 11, 37), new Block(56, 11, 38), new Block(69,
38, 39), new Block(69, 38, 40),
    new Block(25, 25, 41), new Block(25, 25, 42), new Block(35,
45, 43), new Block(35, 45, 44),
    new Block(55, 15, 45), new Block(15, 55, 46), new Block(15,
55, 47), new Block(45, 35, 48),
    new Block(45, 35, 49), new Block(85, 15, 50), new Block(15,
25, 51), new Block(15, 25, 52),
    new Block(15, 25, 53), new Block(15, 25, 54), new Block(35,
25, 55), new Block(35, 25, 56),
    new Block(77, 17, 57), new Block(77, 17, 58), new Block(14,
48, 59), new Block(14, 48, 60),
    new Block(41, 41, 61), new Block(41, 41, 62), new Block(61,
21, 63), new Block(61, 21, 64),
    new Block(22, 22, 65), new Block(22, 22, 66), new Block(32,
32, 67), new Block(32, 32, 68),
    new Block(62, 32, 69), new Block(62, 32, 70), new Block(18,
28, 71), new Block(18, 28, 72),
    new Block(28, 12, 73), new Block(28, 12, 74), new Block(58,
13, 75), new Block(58, 13, 76),
    new Block(71, 36, 77), new Block(71, 36, 78), new Block(44,
44, 79), new Block(44, 44, 80),
    new Block(21, 21, 81), new Block(21, 21, 82), new Block(31,
```

```
31, 83), new Block(31, 31, 84),
    new Block(61, 31, 85), new Block(61, 31, 86), new Block(17,
26, 87), new Block(17, 26, 88),
    new Block(26, 11, 89), new Block(26, 11, 90), new Block(57,
12, 91), new Block(57, 12, 92),
    new Block(70, 35, 93), new Block(70, 35, 94), new Block(43,
43, 95), new Block(43, 43, 96),
    new Block(20, 20, 97), new Block(20, 20, 98), new Block(30,
30, 99), new Block(30, 30, 100)
  ];
  if (n === 20) return base.slice(0, 20);
  if (n === 40) return base.slice(0, 40);
  if (n === 60) return base.slice(0, 60);
  if (n === 80) return base.slice(0, 80);
  if (n === 100) return base;
  return [];
}

//const crossovers = [ 'one_point','arithmetic','ox', 'pmx'];
const crossovers = [ 'arithmetic'];
// Тестова функція для генетичного алгоритму
function runGeneticAlgorithmTests() {
  const blockCounts = [20, 40, 60, 80, 100];
  //const blockCounts = [40];
  //const populationSizes = [1000, 2000, 3000, 4000, 5000];
  const populationSizes = [5000];
  const sheetWidth = 300;
  const sheetHeight = 300;
  const repeats = 5;

  blockCounts.forEach(blockCount => {
    const blocks = getTestBlocks(blockCount);
    populationSizes.forEach(popSize => {
      crossovers.forEach(crossoverType => {
        let totalTime = 0;
        let totalSheets = 0;
        let totalEfficiency = 0;
        for (let rep = 0; rep < repeats; rep++) {
          const startTime = performance.now();
          let resSheets = runGeneticAlgorithm(
            popSize, 0.01, popSize, blocks, sheetWidth,
sheetHeight, undefined, crossoverType
          );
          const endTime = performance.now();
          const executionTime = endTime - startTime;
          totalTime += executionTime;
          totalSheets += resSheets.length;
          totalEfficiency +=
parseFloat(calculateEfficiency(resSheets));
        }
        const avgTime = totalTime / repeats;
        const avgSheets = totalSheets / repeats;
        const avgEfficiency = totalEfficiency / repeats;
```

```
        console.log('='.repeat(60));
        console.log(`TEST: Blocks=${blockCount},
Population=${popSize}, Crossover=${crossoverType}`);
        console.log('-'.repeat(60));
        console.log(`Average Execution Time:
${avgTime.toFixed(2)} ms`);
        console.log(`Average Number of Sheets:
${avgSheets.toFixed(2)}`);
        console.log(`Average Efficiency:
${avgEfficiency.toFixed(2)}`);
        console.log('='.repeat(60) + '\n');
      });
    });
  });
}

function calculateEfficiency(sheets) {
  let minX = Infinity, minY = Infinity, maxX = 0, maxY = 0;
  sheets.forEach(sheet => {
    sheet.blocks.forEach(block => {
      minX = Math.min(minX, block.x);
      minY = Math.min(minY, block.y);
      maxX = Math.max(maxX, block.x + (block.w ? block.w :
block.block.w));
      maxY = Math.max(maxY, block.y + (block.h ? block.h :
block.block.h));
    });
  });
  let totalArea = (maxX - minX) * (maxY - minY);
  let usedArea = sheets.reduce((sum, sheet) => {
    return sum + sheet.blocks.reduce((blockSum, block) => {
      return blockSum + (block.w ? block.w * block.h :
block.block.w * block.block.h);
    }, 0);
  }, 0);
  let efficiency = (usedArea / totalArea) * 100 / sheets.length;
  return efficiency.toFixed(2);
}

// Тестова функція для мурашиного алгоритма (ACO)
function runACOTests() {
  // Импортируйте runACO, если нужно: import { runACO } from
'../Model/antColonyAlgorithm.js';
  //const blockCounts = [20, 40, 60, 80, 100];
  const blockCounts = [20, 40, 60, 80, 100];
  const antCounts = [100, 300, 500, 1000];
  //const iterationsArr = [1000, 2000, 3000, 4000];
  const iterationsArr = [1000];
  const sheetWidth = 300;
  const sheetHeight = 300;
  const repeats = 5;

  blockCounts.forEach(blockCount => {
```

```
    const blocks = getTestBlocks(blockCount);
    antCounts.forEach(antCount => {
      iterationsArr.forEach(iterations => {
        let totalTime = 0;
        let totalSheets = 0;
        let totalEfficiency = 0;
        for (let rep = 0; rep < repeats; rep++) {
          const startTime = performance.now();
          let resSheets = runACO(
            blocks, sheetWidth, sheetHeight,
            antCount, iterations, 1, 2, 0.5
          );
          const endTime = performance.now();
          const executionTime = endTime - startTime;
          totalTime += executionTime;
          totalSheets += resSheets.length;
          totalEfficiency +=
parseFloat(calculateEfficiency(resSheets));
        }
        const avgTime = totalTime / repeats;
        const avgSheets = totalSheets / repeats;
        const avgEfficiency = totalEfficiency / repeats;
        console.log('='.repeat(60));
        console.log(`ACO TEST: Blocks=${blockCount},
Ants=${antCount}, Iterations=${iterations}`);
        console.log('-'.repeat(60));
        console.log(`Average Execution Time:
${avgTime.toFixed(2)} ms`);
        console.log(`Average Number of Sheets:
${avgSheets.toFixed(2)}`);
        console.log(`Average Efficiency:
${avgEfficiency.toFixed(2)}`);
        console.log('='.repeat(60) + '\n');
      });
    });
  });
}

// Тестова функція для симуляції отжигу (Simulated Annealing)
function runSimulatedAnnealingTests() {
  const blockCounts = [20, 40, 60, 80, 100];
  const initialTemperatures = [100];
  const alpha = 0.5;
  const maxIterationsArr = [1e6, 1e7, 1e8, 1e9];
  const sheetWidth = 300;
  const sheetHeight = 300;
  const repeats = 5;
  const maxIterations = 1000

  blockCounts.forEach(blockCount => {
    const blocks = getTestBlocks(blockCount);
    initialTemperatures.forEach(initialTemperature => {
        //maxIterationsArr.forEach(maxIterations => {
```

```
          let totalTime = 0;
          let totalSheets = 0;
          let totalEfficiency = 0;
          for (let rep = 0; rep < repeats; rep++) {
            const startTime = performance.now();
            let resSheets = runSimulatedAnnealing(
              blocks, sheetWidth, sheetHeight,
              {
                initialTemperature,
                alpha,
                maxIterations
              }
            );
            const endTime = performance.now();
            const executionTime = endTime - startTime;
            totalTime += executionTime;
            totalSheets += resSheets.length;
            totalEfficiency +=
parseFloat(calculateEfficiency(resSheets));
          }
          const avgTime = totalTime / repeats;
          const avgSheets = totalSheets / repeats;
          const avgEfficiency = totalEfficiency / repeats;
          console.log('='.repeat(60));
          console.log(`SA TEST: Blocks=${blockCount},
T0=${initialTemperature}, alpha=${alpha},
maxIter=${maxIterations}`);
          console.log('-'.repeat(60));
          console.log(`Average Execution Time:
${avgTime.toFixed(2)} ms`);
          console.log(`Average Number of Sheets:
${avgSheets.toFixed(2)}`);
          console.log(`Average Efficiency:
${avgEfficiency.toFixed(2)}`);
          console.log('='.repeat(60) + '\n');
        });
    //});
  });
}

// Тестова функція для алгоритма Best Fit
function runBestFitTests() {
  const blockCounts = [20, 40, 60, 80, 100];
  const sheetWidth = 300;
  const sheetHeight = 300;
  const repeats = 5;

  blockCounts.forEach(blockCount => {
    const blocks = getTestBlocks(blockCount);
    let totalTime = 0;
    let totalSheets = 0;
    let totalEfficiency = 0;
    for (let rep = 0; rep < repeats; rep++) {
```

```
      const startTime = performance.now();
      let resSheets = BestFit.cutBlocks(blocks, sheetWidth,
sheetHeight);
      const endTime = performance.now();
      const executionTime = endTime - startTime;
      totalTime += executionTime;
      totalSheets += resSheets.length;
      totalEfficiency +=
parseFloat(calculateEfficiency(resSheets));
    }
    const avgTime = totalTime / repeats;
    const avgSheets = totalSheets / repeats;
    const avgEfficiency = totalEfficiency / repeats;
    console.log('='.repeat(60));
    console.log(`BEST FIT TEST: Blocks=${blockCount}`);
    console.log('-'.repeat(60));
    console.log(`Average Execution Time: ${avgTime.toFixed(2)}
ms`);
    console.log(`Average Number of Sheets:
${avgSheets.toFixed(2)}`);
    console.log(`Average Efficiency:
${avgEfficiency.toFixed(2)}`);
    console.log('='.repeat(60) + '\n');
  });
}

// Тестова функція для гібридного алгоритма (Hybrid Genetic)
function runHybridGeneticAlgorithmTests() {
  const blockCounts = [20, 40, 60, 80, 100];
  const populationSizes = [5000];
  const sheetWidth = 300;
  const sheetHeight = 300;
  const repeats = 5;
  const crossoverType = 'arithmetic';

  blockCounts.forEach(blockCount => {
    const blocks = getTestBlocks(blockCount);
    populationSizes.forEach(popSize => {
      let totalTime = 0;
      let totalSheets = 0;
      let totalEfficiency = 0;
      for (let rep = 0; rep < repeats; rep++) {
        const startTime = performance.now();
        let resSheets = runHybridGeneticAlgorithm(
          popSize, 0.01, popSize, blocks, sheetWidth,
sheetHeight,  crossoverType
        );
        const endTime = performance.now();
        const executionTime = endTime - startTime;
        totalTime += executionTime;
        totalSheets += resSheets.length;
        totalEfficiency +=
parseFloat(calculateEfficiency(resSheets));
```

```
      }
      const avgTime = totalTime / repeats;
      const avgSheets = totalSheets / repeats;
      const avgEfficiency = totalEfficiency / repeats;
      console.log('='.repeat(60));
      console.log(`HYBRID GENETIC TEST: Blocks=${blockCount},
Population=${popSize}`);
      console.log('-'.repeat(60));
      console.log(`Average Execution Time: ${avgTime.toFixed(2)}
ms`);
      console.log(`Average Number of Sheets:
${avgSheets.toFixed(2)}`);
      console.log(`Average Efficiency:
${avgEfficiency.toFixed(2)}`);
      console.log('='.repeat(60) + '\n');
    });
  });
}

$(Demo.init);

//runSimulatedAnnealingTests();
//runACOTests();
//runGeneticAlgorithmTests();
//runBestFitTests();
//runBestFitTests();
//runHybridGeneticAlgorithmTests()
```

Б.4 Код сторінки та її стилізація

Б.4.1 Файл index.html

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset='utf-8' />
  <title>Bin Packing</title>
  <link href='../View/demo.css' rel='stylesheet' type='text/css'
media='screen' />
</head>
<body>

  <h1>Розкрій листового матеріалу</h1>

  <div id="container">
    <fieldset id="settings">
      <legend>Налаштування</legend>

      <div class='size'>
```

```
        <label for='size' class="blockLabel">Розмір листа (см)
<span class='example'><b>W</b>x<b>H</b></span></label>
        <br>
        <input id="size_w" value="" size="5"><ul id="dropdownSW"
class="dropdown-list"></ul>x<input id="size_h"  value=""
size="5"><ul id="dropdownSH" class="dropdown-list"></ul><div
class="size_report" id="size_report">Обмеження до 3000см!</div>
      </div>

      <div class='blocks'>
        <label for='blocks'>Деталі (см) <span
class='example'><b>W</b>x<b>H</b>x<b>N</b></span></label>
        <br>
        <input id="block_w" value="" size="5"><ul id="dropdownW"
class="dropdown-list"></ul>x<input id="block_h"  value=""
size="5"><ul id="dropdownH" class="dropdown-list"></ul>x<input
id="block_n"  value="" size="5"><ul id="dropdownN"
class="dropdown-list"></ul>
        <button name="add_block" id="add_block">Додати
деталь</button>
        <label class="toggle-switch">
          <input type="checkbox" id="toggle_details">
          <span class="slider"></span>
        </label>
        <div id="details" class="details"></div>
        <div>
      <label for="algorithm_select">Виберіть алгоритм:</label>
      <select id="algorithm_select">
        <option value="best_fit">Best Fit</option>
        <option value="genetic">Genetic</option>
        <option value="ant_colony">Ant colony</option>
        <option value="simulated_annealing">Simulated
annealing</option>
        <option value="hybrid_genetic">Hybrid Genetic</option>
      </select>
    </div>
      <button name="run_button" id="run_button">Запустити
алгоритм</button>
      <br>
      <div id="execution_time" class="info-box">Час виконання: 0
мс</div>
      <div id="efficiency" class="info-box">Оцінка: 0%</div>
      <div id="numSheets" class="info-box">Кількість листів: 0
шт</div>
        <button name="edit_block" id="edit_block" >Редагувати д.
№</button>
        <input id="edit_block_n" value="" size="1">
        <input id="edit_block_w" value="" size="1">x<input
id="edit_block_h" value="" size="1">
      </div>


      <div id="numSheets" class="numSheets"></div>
```

```
      <button name="saveToPdf" id="saveToPdf"
onclick="canvas.saveToPdf()">Зберегти результат у PDF</button>
      <span id="definitions">*W - широта<br>*H - висота<br> *N -
кількість</span>
    </fieldset>

      <div id="packing">
        <canvas id="canvas">
          <div id="unsupported">
            Sorry, this example cannot be run because your
browser does not support the &lt;canvas&gt; element
          </div>
        </canvas>
      </div>
  </div>

  <script src='../Libraries/jquery.js'></script>
  <script src='../Libraries/jspdf.min.js'></script>
  <script src='../Controller/demo.js'></script>
  <script src='../Model/Block.js'></script>
  <script src='../Model/Sheet.js'></script>
  <script src='../View/canvas.js'></script>
  <script src='../Model/details.js'></script>
  <script src='../Model/BestFit.js'></script>
  <script src='../Model/antColonyAlgorithm.js'></script>
  <script src='../Model/simulatedAnnealing.js'></script>
  <script src='../Model/hybridGeneticAlgorithm.js'></script>
  <script src='../Model/geneticAlgorithm.js'></script>
</body>
</html>
```

Б.4.2 CSS файл

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Inter', sans-serif;
}

body {
    background: #b3e5fc2e;
    display: flex;
    padding: 1rem 2rem;
    color: #333;
    min-height: 100vh;
    flex-direction: column;
    align-items: center;
    position: relative;
}
```

```
h1 {
    text-align: center;
    width: 100%;
    margin-top: 0;
    margin-bottom: 1.5rem;
    font-size: 2.2rem;
    font-weight: 800;
    letter-spacing: 0.02em;
    color: #1a1a1a;
    line-height: 1.1;
    background: transparent;
    pointer-events: auto;
}

/* Layout */
#container {
    display: flex;
    width: 100%;
    max-width: 1400px;
    margin: 0 auto;
    padding: 0;
    gap: 2em;
}

.sidebar, #settings {
    width: 300px;
    min-width: 300px;
    padding: 1.1rem 1.1rem 0.7rem 1.1rem;
    background: #fff;
    border-radius: 9px;
    box-shadow: 0 4px 10px rgba(0,0,0,0.17);
    margin: 0;
    display: block;
    float: none;
    border: none;
}

#settings legend,
.sidebar h2 {
    font-size: 1.23rem;
    color: #222;
    font-weight: 700;
    background: none;
    border: none;
    padding: 0;
    text-align: center;
    display: block;
}

label {
    font-weight: 600;
    color: #222;
    font-size: 0.97em;
```

```
}

#size_w {
    margin-bottom: 12px;
}

select {
    width: 98%;
    display: inline-block;
    padding: 0.32rem;
    border: 1px solid #ccc;
    border-radius: 6px;
    font-size: 0.97em;
    background: #fff;
    color: #222;
    outline: none;
    transition: border 0.2s;
    margin-top: 12px;
}

input {
    width: 22%;
    display: inline-block;
    padding: 0.32rem;
    margin-top: 0.7rem;
    border: 1px solid #ccc;
    border-radius: 6px;
    font-size: 0.97em;
    background: #fff;
    color: #222;
    outline: none;
    transition: border 0.2s;
}

input:focus, select:focus {
    border: 1.5px solid #007bff;
}

button {
    width: 100%;
    padding: 0.55rem;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 6px;
    font-weight: 600;
    cursor: pointer;
    font-size: 0.97em;
    transition: background 0.18s, transform 0.18s;
    margin-top: 12px;
}

button:hover, button:focus {
```

```
    background: #0056b3;
    transform: translateY(-2px) scale(1.02);
}

.info-box {
    padding: 0.55rem;
    border: 1px solid #ccc;
    border-radius: 6px;
    background: #f0f0f0;
    text-align: center;
    margin-top: 0.7rem;
    color: #222;
    font-weight: 500;
    font-size: 0.96em;
}

.size_report {
    display: none;
    border: 1px solid #ffbdbd;
    background: #fff6f6;
    color: #b30000;
    margin-bottom: 0.7rem;
    padding: 0.5em 0.7em;
    border-radius: 5px;
    font-size: 0.95em;
}


.details.hidden {
    display: none;
}

#definitions {
    color: #8b8b8b;
    font-style: italic;
    font-size: 0.93em;
    margin-top: 0.4rem;
    display: block;
}

.ratio, #settings .ratio {
    background: #f0f0f0;
    border-radius: 6px;
    padding: 0.5rem 0.7rem;
    font-size: 0.96em;
    color: #222;
    font-weight: 500;
    margin-bottom: 0.7rem;
    border: 1px solid #e0e0e0;
}

.toggle-switch {
    position: relative;
```

```
    display: inline-block;
    width: 32px;
    height: 18px;
    vertical-align: middle;
    margin: 0.7rem 0;
}

.toggle-switch input {
    opacity: 0;
    width: 0;
    height: 0;
}

.slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: #e0e0e0;
    transition: .3s;
    border-radius: 22px;
}

.slider:before {
    position: absolute;
    content: "";
    height: 12px;
    width: 12px;
    left: 2px;
    bottom: 3px;
    background: #fff;
    transition: .3s;
    border-radius: 50%;
    box-shadow: 0 1px 3px rgba(0,0,0,0.07);
}

input:checked + .slider {
    background: #007bff;
}

input:checked + .slider:before {
    transform: translateX(12px);
}

.dropdown-list {
    display: none;
    position: absolute;
    list-style: none;
    margin: 0;
    padding: 0;
    background: #fff;
```

```
    border: 1px solid #e0e0e0;
    border-radius: 5px;
    box-shadow: 0 2px 8px rgba(0,0,0,0.07);
    z-index: 9999;
    min-width: 48px;
}

.dropdown-list li {
    color: #222;
    padding: 5px 10px;
    cursor: pointer;
    font-size: 0.96em;
    border-bottom: 1px solid #f2f2f2;
    transition: background 0.15s;
}

.dropdown-list li:last-child {
    border-bottom: none;
}

.dropdown-list li:hover {
    background: #f5f5f5;
}

#dropdownH, #dropdownSH { left: 150px; }
#dropdownN { left: 230px; }

.delButton {
        color: #fff;
    font-size: 9px;
    font-weight: bold;
    display: inline-block;
    margin: 0 0 4px 4px;
    background: #b30000;
    width: 35px;
    height: 35px;
    text-align: center;
    cursor: pointer;
    border: none;
    transition: background 0.2s;
}

.delButton:hover {
    background: #ff3b3b;
}

.main, #results {
    flex-grow: 1;
    display: flex;
    align-items: flex-start;
    justify-content: flex-start;
}
```

```
#packing {
    margin: 0px 20px;
    padding: 0;
    width: 50%;
    min-width: 320px;
    display: flex;
    align-items: flex-start;
    justify-content: flex-start;
}

#canvas {
    border: 2px solid #ddd;
    border-radius: 9px;
    background: #fff;
    display: block;
    margin: 0 auto;
}

#unsupported {
    border: none;
    background: #ffeaea;
    color: #b30000;
    padding: 1.2em;
    margin: 1em;
    border-radius: 8px;
    font-size: 1.1em;
}

.block {
    background-color: #b3e5fc;
    border: 1px solid #90caf9;
    border-radius: 4px;
    box-shadow: 0 1px 3px rgba(0,0,0,0.1);
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 12px;
}

.input-row {
    display: flex;
    gap: 2.5%;
    margin-bottom: 0.7rem;
}

.input-row input,
.input-row select {
    width: 100%;
    margin-bottom: 0;
}
```