УДК 65.011.56

АВТОМАТИЗАЦИЯ ПОСТРОЕНИЯ СИСТЕМ ПРОИЗВОДСТВЕННОГО ПЛАНИРОВАНИЯ

Усынин Андрей Вячеславових, студент, Базовая кафедра «Аналитика больших данных и методы видеоанализа», Уральский федеральный университет имени первого Президента России Б.Н. Ельцина

Машанов Илья Владимирович, студент, Базовая кафедра «Аналитика больших данных и методы видеоанализа», Уральский федеральный университет имени первого Президента России Б.Н. Ельцина

Дударев Дмитрий Константинович, студент, Базовая кафедра «Аналитика больших данных и методы видеоанализа», Уральский федеральный университет имени первого Президента России Б.Н. Ельцина

Медведев Максим Александрович. к.т.н., доцент Базовой кафедры «Аналитика больших данных и методы видеоанализа», Уральский федеральный университет имени первого Президента России Б.Н. Ельцина

Аннотация

В работе представлен инструмент для автоматизации построения систем производственного планирования, который предназначен для преобразования условий задач математического программирования, описанных в табличном фреймворке Univer, в исполняемый Python-код с использованием библиотеки PuLP. Разработка и реализация данного инструмента решает актуальную проблему, связанную с необходимостью ручного переноса моделей, разработанных аналитиками, в промышленный программный код, что часто приводит к потерям времени, неточностям и увеличению затрат на разработку. Предложенное решение обеспечивает автоматизированный и безошибочный переход от аналитических прототипов к промышленной реализации, сокращая сроки внедрения и повышая эффективность работы команд разработки и аналитиков. В результате автоматизации снижаются издержки на разработку, уменьшается вероятность возникновения ошибок и существенно ускоряется цикл внедрения математических моделей оптимизации в производственную практику.

КЛЮЧЕВЫЕ СЛОВА: Python, PuLP, Excel, Univer, TypeScript, генерация кода, математическое программирование, линейное программирование, цифровизация производства, автоматизация

COMPUTER TOOL FOR AUTOMATION OF DESIGNING OF PRODUCTION PLANNING SYSTEMS

Usynin Andrey Vyacheslavovich, Student, Department of Big Data Analytics and Video Analysis Methods, Ural Federal University named after the first President of Russia B.N. Yeltsin

Mashanov Ilya Vladimirovich, Student, Department of Big Data Analytics and Video Analysis Methods, Ural Federal University named after the first President of Russia B.N. Yeltsin

Dudarev Dmitriy Konstantinovich, Student, Department of Big Data Analytics and Video Analysis Methods, Ural Federal University named after the first President of Russia B.N. Yeltsin

Medvedev Maksim Aleksandrovich, Associate Professor, Department of Big Data Analytics and Video Analysis Methods, Ural Federal University named after the first President of Russia B.N. Yeltsin

Abstract

This paper presents a tool for automating the construction of production planning systems, which is designed to convert the conditions of mathematical programming problems described in the Univer table framework into executable Python code using the PuLP library. The implementation of this tool solves the actual problem associated with the need to manually transfer models developed by analysts into industrial code, which often leads to lost time, inaccuracies and increased development costs. The proposed solution provides an automated and error-free transition from analytical prototypes to production implementation, reducing implementation time and increasing the efficiency of development teams and analysts. As a result of

automation, development costs are reduced, the probability of errors is reduced, and the cycle of implementation of mathematical optimization models into production practice is significantly accelerated.

KEYWORDS: Python, PuLP, Excel, Univer, TypeScript, code generation, mathematical programming, linear programming, digitalization of manufacturing, automation.

1. Введение

Современные производственные предприятия вынуждены постоянно повышать эффективность управления ресурсами и процессами в условиях растущей конкуренции и сложных рыночных условий. Одним из важнейших инструментов, способных обеспечить повышение эффективности, является математическое программирование, позволяющее оптимизировать процессы принятия решений на различных уровнях производственной деятельности.[1, 2, 3, 4] Однако процесс создания, тестирования и внедрения математических моделей остается трудоемким и затратным [5]. Аналитики, ответственные за первичное моделирование и прототипирование задач, чаще всего используют доступные и удобные инструменты, такие как Excel или аналогичные табличные редакторы [6]. Эти инструменты идеально подходят для быстрого создания и проверки гипотез на небольших объемах данных, однако практически не пригодны для интеграции с промышленными системами управления.

В результате возникает необходимость повторного ручного переноса уже проверенных и работоспособных моделей аналитиков в производственный код, используемый разработчиками в промышленных информационных системах. Данный перенос сопряжен с рядом негативных факторов: возрастают временные и финансовые затраты, увеличивается вероятность ошибок из-за человеческого фактора, усложняется процесс оперативного внесения изменений и модификаций в модель, что критически важно в современных условиях быстрого изменения рыночной ситуации.

Исследование направлено на решение указанной проблемы путем создания автоматизированного инструмента, интегрированного в аналитическую платформу Univer, который способен самостоятельно анализировать модели математического программирования, описанные в табличном виде, и преобразовывать их в промышленный код на языке Python с использованием специализированной библиотеки PuLP. Внедрение такого инструмента существенно сократит временные промежутки между этапами прототипирования аналитиками и внедрения моделей разработчиками, что приведет к экономии ресурсов предприятия и повышению оперативности и точности принимаемых решений.

В работе проводится исследование архитектуры предлагаемого решения, детально рассматриваются используемые технологии и подходы, определяется влияние автоматизированного инструмента на производственные процессы. Кроме того, оцениваются перспективы дальнейшего развития решения, в том числе расширение функциональности и интеграция с другими распространенными библиотеками и инструментами.

2. Материалы и методы

2.1. Анализ структуры и требований задач производственного планирования

Разработка инструмента была начата с анализа существующих практик решения задач производственного планирования, используемых аналитиками компании - заказчика. Основное внимание уделялось изучению подходов аналитиков к решению задач линейного и целочисленного математического программирования с применением табличных редакторов, таких как Univer и Excel. Были изучены и описаны типовые элементы задач: целевые функции, системы ограничений и используемые переменные. Анализ позволил выявить ключевые трудности, возникающие при переносе задач из табличных форматов в промышленный код, а именно ручное преобразование данных, риск ошибок и значительные временные затраты специалистов (аналитиков и разработчиков) [7, 8, 9, 10].

На основе проведенного анализа были определены основные функциональные требования к разрабатываемому инструменту, в число которых вошли автоматизированный парсинг исходных данных, генерация исполняемого кода на языке Python с использованием специализированной библиотеки PuLP, а также интеграция инструмента непосредственно в рабочее окружение аналитиков посредством пользовательского интерфейса редактора Univer.

2.2. Разработка плагина для Univer

Основой для реализации инструмента была выбрана библиотека PuLP языка Python, широко используемая для решения задач математического программирования благодаря простоте использования и эффективности. Разработка программного инструмента была разделена на две основные части: создание парсера данных и разработку генератора Python-кода.

Для работы с фреймворком Univer есть 2 пути:

1. Извне через Facade API, как с обычной библиотекой, из-за чего невозможно вписаться напрямую в интерфейсе таблиц Univer и соответственно появляется необходимость в добавлении интерфейса за его пределами.
2. Написание плагина, имеющего полный доступ к Univer, так как он интегрируется напрямую в таблицу.

В результате выбора второго подхода появилась возможность добавления интерфейса взаимодействия непосредственно в Univer (Рисунок 1).

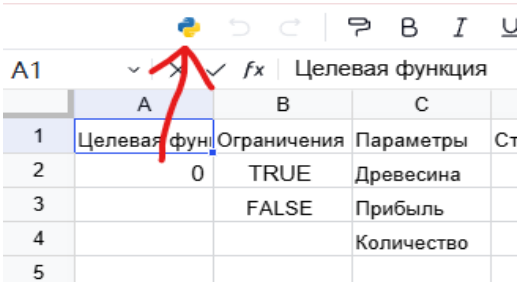

Рис. 1. Кнопка взаимодействия с плагином

Структура плагина представлена в следующем виде:

```
src/
├── App.tsx
└── plugin/
    ├── plugin.ts
    ├── controllers/
    │   └── export-button.controller.ts
    ├── commands/
    │   └── operations/
    │       └── export-button.operation.ts
    ├── components/
    │   └── icons/
    │       └── python-icon.ts
    └── locales/
        ├── ru-RU.ts
        └── en-US.ts
```

Основной файл плагина plugin.ts состоит исключительно из объявления контроллера и добавления файлов локализации (Рисунок 2).

```
export class UniverSheetsPythonExportPlugin extends Plugin {
  static override type = UniverInstanceType.UNIVER_SHEET;
  static override pluginName = SHEET_CUSTOM_MENU_PLUGIN;

  constructor(
    @Inject(Injector) protected readonly _injector: Injector,
    @Inject(LocaleService) private readonly _localeService: LocaleService
  ) {
    super();

    this._localeService.load({
      ruRU,
      enUS,
    });
  }

  override onStarting(): void {
    ([[ExportButtonController]] as Dependency[]).forEach((d) =>
      this._injector.add(d)
    );
  }

  onReady(): void {
    this._injector.get(ExportButtonController);
  }
}
```

Рис. 2. Код класса плагина

Именно этот файл подключается при инициализации Univer Spreadsheets в App.tsx (Рисунок 3).

```
const { univerAPI } = createUniver({
  locale: LocaleType.RU_RU,
  locales: {
    [LocaleType.RU_RU]: merge({}, UniverPresetSheetsCoreRuRU),
    [LocaleType.EN_US]: merge({}, UniverPresetSheetsCoreEnUS),
  },
  theme: defaultTheme,
  presets: [
    UniverSheetsCorePreset({
      container: containerRef.current ?? undefined,
    }),
  ],
  plugins: [UniverSheetsPythonExportPlugin],
});
```

Рис. 3. Создание объекта Univer Spreadsheets

Класс контроллера обеспечивает регистрацию команды экспорта (основной логики плагина, то есть генерации Python кода и экспорта в файл), добавления иконки и самой кнопки в меню Univer (Рисунок 4).

```
/**
 * register commands
 */
private _initCommands(): void {
  [ExportButtonOperation].forEach((c) => {
    this.disposeWithMe(this._commandService.registerCommand(c));
  });
}

/**
 * register icon components
 */
private _registerComponents(): void {
  const componentManager = this._componentManager;
  this.disposeWithMe(componentManager.register("PythonIcon", PythonIcon));
}

/**
 * register menu items
 */
private _initMenus(): void {
  this._menuMangerService.mergeMenu({
    [RibbonStartGroup.HISTORY]: {
      [ExportButtonOperation.id]: {
        order: -1,
        menuItemFactory: ExportButtonFactory,
      },
    },
  });
}
```

Рис. 4. Логика класса контроллера

Команда экспорта получает объект сервиса Univer с помощью механизма Dependency Injection и из него забирает текущий редактируемый лист в таблице (Рисунок 5).

```
export const ExportButtonOperation: ICommand = {
  id: "export-button.operation.export-button",
  type: CommandType.OPERATION,
  handler: async (accessor: IAccessor) => {
    const univer = accessor.get(IUniverInstanceService);
    const sheet = univer
      .getCurrentUnitOfType<Workbook>(UniverInstanceType.UNIVER_SHEET)!
      .getActiveSheet();
    exportPython(sheet);
    return true;
  },
};
```

Рис. 5. Код команды экспорта

Перед запуском генерации кода Python, в функции exportPython происходит получение и валидация данных таблицы (Рисунок 6), а именно целевая функция из ячейки A1, ограничения из столбца B и все необходимые для генерации переменные. В случае, если переменная не заполнена в соответствующей ячейке таблицы, она будет считаться искомой.

```
const exportPython = (worksheet: Worksheet) => {
  try {
    const sheetSnapshot = worksheet.getSnapshot();
    if (
      !Object.keys(sheetSnapshot.cellData).includes("1") ||
      !Object.keys(sheetSnapshot.cellData[1]).includes("0")
    ) {
      alert("Укажите целевую функцию в ячейке A1");
      return;
    }
    const func = sheetSnapshot.cellData[1][0].f?.trim();
    console.log("Целевая функция", func);
    if (!func) {
      alert("Укажите целевую функцию в ячейке A1");
      return;
    }
    const restrictions = [];
    for (const [, row] of Object.entries(sheetSnapshot.cellData)) {
      const cell = row[1];
      if (cell && cell.f) restrictions.push(cell.f.trim());
    }
    console.log("Ограничения", restrictions);
    const variables = extractVariables([func, ...restrictions]).map((v) =>
      v.trim()
    );
    console.log("Переменные", variables);
    const varData = variables.reduce<Record<string, string>>(
      (res, variable) => {
        const row = A1.getRow(variable) - 1;
        const col = A1.getCol(variable) - 1;
        if (
          !Object.keys(sheetSnapshot.cellData).includes(row.toString()) ||
          !Object.keys(sheetSnapshot.cellData[row]).includes(col.toString())
        )
          return res;
        const val = sheetSnapshot.cellData[row][col].v?.toString().trim();
        if (!val) return res;
        return { [variable]: val, ...res };
      },
      {}
    );
    console.log("Переменные", varData);
    const pythonCode = Convert(func, restrictions, varData);
    download(pythonCode);
  } catch (ex) {
    alert(
      "Во время конвертации произошла ошибка, проверьте правильность формул"
    );
    throw ex;
  }
};
```

Рис. 6. Код получения и валидации данных таблицы

После конвертации происходит скачивание файла с результатом в виде Python кода (Рисунок 7).

```
const download = (text: string) => {
  const element = document.createElement("a");
  element.setAttribute(
    "href",
    "data:text/plain;charset=utf-8," + encodeURIComponent(text)
  );
  element.setAttribute("download", "code.py");

  element.style.display = "none";
  document.body.appendChild(element);

  element.click();

  document.body.removeChild(element);
};
```

Рис. 7. Код скачивания итогового файла

2.3. Составление шаблона для генерации Python файлов

Результатом работы плагина является Python код. Пример представлен на Рисунке 8.

```python
import pulp

# Define known variables
x_E2 = 3
x_D2 = 4
x_E3 = 200
x_D3 = 300

# Define optimization variables
x_D4 = pulp.LpVariable('D4', lowBound=0)
x_E4 = pulp.LpVariable('E4', lowBound=0)

# Define problem
prob = pulp.LpProblem('LP_Problem', pulp.LpMaximize)

# Objective
prob += (x_D3*x_D4+x_E3*x_E4), 'Objective'

# Constraints
prob += (x_D2*x_D4+x_E2*x_E4 <= 10), 'Constraint_1'
prob += (x_D4+x_E4 >= 3), 'Constraint_2'

# Solve
prob.solve()

print("Status:", pulp.LpStatus[prob.status])
for v in prob.variables():
    print(f"{v.name} = {v.varValue}")
print("Objective =", pulp.value(prob.objective))
```

Рис. 8. Код Python файла

Для работы инструмента необходимо установить Python и библиотеку PuLP:

‘pip install pulp’ либо *‘python3 -m pip install pulp’*

Чтобы запустить код в терминале выполните команду:

‘python code.py’ (в той же папке, где файл).

Здесь *x_D3* и *x_E3* — коэффициенты при переменных в целевой функции.

x_D2 и *x_E2* — коэффициенты в ограничениях.

Далее задаются неизвестные переменные *x_D4* и *x_E4*, которые оптимизируются. *lowBound=0* означает, что переменные не могут быть отрицательными.

prob = pulp.LpProblem('LP_Problem', pulp.LpMaximize) – создается задача с названием 'LP_Problem'. LpMaximize обозначает то, что значение целевой функции максимизируется. Для минимизации соответственно LpMinimize.

*prob += (x_D3*x_D4 + x_E3*x_E4), 'Objective'* - основная функция, которую нужно максимизировать.

*prob += (x_D2*x_D4 + x_E2*x_E4 <= 10), 'Constraint_1'* и

prob += (x_D4 + x_E4 >= 3), 'Constraint_2' – ограничения, которые задают допустимые значения переменных.

prob.solve() – вызов метода решения задачи.

LpStatus[prob.status] — текстовое описание результата (например, 'Optimal').

prob.variables() — список оптимальных значений переменных.

pulp.value(prob.objective) — значение целевой функции при найденном решении.

```
At line 2 NAME          MODEL
At line 3 ROWS
At line 7 COLUMNS
At line 14 RHS
At line 17 BOUNDS
At line 18 ENDATA
Problem MODEL has 2 rows, 2 columns and 4 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 0 (-2) rows, 0 (-2) columns and 0 (-4) elements
Empty problem - 0 rows, 0 columns and 0 elements
Optimal - objective value 700
After Postsolve, objective 700, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 700 - 0 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.00

Status: Optimal
D4 = 1.0
E4 = 2.0
Objective = 700.0
```

Рис. 9. Результат выполнения Python файла

Это означает, что при D4 = 1 и E4 = 2 достигается максимум целевой функции 700 с соблюдением всех ограничений.

2.4. Интеграция инструмента в среду Univer и тестирование

Для обеспечения удобства работы пользователей была разработана система интеграции созданного инструмента непосредственно в интерфейс табличного редактора Univer посредством специализированного плагина на языке TypeScript. Плагин интегрирует в интерфейс Univer кнопку «Экспорт в Python», инициирующую весь процесс преобразования данных таблицы в исполняемый код. После нажатия кнопки пользователем запускается процедура извлечения и валидации данных, генерация кода и последующая загрузка готового Python-файла на устройство пользователя.

Тестирование инструмента проводилось на типичных задачах производственного планирования разного уровня сложности. В ходе тестирования были выявлены и исправлены ошибки в обработке данных, улучшена отказоустойчивость системы, а также оптимизирована процедура генерации кода. В результате тестирования была подтверждена работоспособность инструмента, его удобство для аналитиков, а также доказано значительное сокращение

времени, необходимого для переноса задач из стадии прототипирования в промышленную эксплуатацию.

Подробное описание процесса установки и настройки плагина, включая все необходимые зависимости и инструкции по интеграции в интерфейс табличного редактора Univer, было приведено в файле README, размещенном в репозитории проекта на платформе GitHub (https://github.com/AnemonOFF/univer-to-py).

3. Результаты и обсуждение

3.1. Цифровой двойник с объектными отношениями

Разработанный инструмент для автоматического построения систем производственного планирования демонстрирует значительные преимущества с точки зрения повышения эффективности и сокращения временных затрат на реализацию математических моделей [6]. Внедрение данного подхода устраняет ключевую проблему, связанную с ручным переносом задач из аналитической среды в промышленную реализацию, минимизируя вероятность человеческих ошибок и сокращая время между этапами проектирования и внедрения готовых решений. Это особенно важно в условиях современного динамичного производства, где оперативность реакции на изменения и качество принимаемых решений непосредственно влияют на конкурентоспособность предприятий.

Одной из главных сильных сторон разработанного подхода является интеграция инструмента непосредственно в рабочий процесс аналитиков через плагин к табличному редактору Univer. Пользователи получают возможность напрямую взаимодействовать с инструментом, не покидая привычной среды работы, что обеспечивает высокую степень комфорта и снижает порог входа для новых пользователей.

Однако следует учитывать ряд ограничений и потенциальных трудностей, которые могут возникнуть при использовании данного подхода. Прежде всего, качество и корректность работы инструмента во многом зависят от структуры и полноты исходных данных, предоставляемых аналитиками. Несмотря на реализованный механизм валидации данных, в случае их недостаточной стандартизации или некорректного форматирования могут возникать ошибки, требующие ручного вмешательства. Таким образом, важной задачей для дальнейшего улучшения инструмента становится разработка более гибких и интеллектуальных алгоритмов предварительной обработки и проверки данных.

Еще одним аспектом, требующим дальнейшего внимания, является адаптация инструмента к более сложным моделям производственного планирования, в которых используются нелинейные ограничения или динамические зависимости между переменными.

Хотя библиотека PuLP отлично подходит для большинства задач линейного и целочисленного программирования, она не всегда способна эффективно обрабатывать сложные нелинейные сценарии. В перспективе это может потребовать интеграции дополнительных библиотек и разработки специализированных методов анализа и генерации кода.

Несмотря на эти ограничения, предложенный инструмент демонстрирует высокий потенциал для дальнейшего развития. Перспективным направлением является также расширение интерфейсной составляющей, включая более глубокую интеграцию с системами управления предприятием (ERP, MES), что позволит получать исходные данные напрямую из промышленных систем, минуя этап ручного ввода. Кроме того, интеграция с платформами для анализа больших данных и искусственного интеллекта может расширить возможности инструмента за счет интеллектуального анализа и прогнозирования производственных процессов.

Таким образом, разработанный инструмент не только решает насущные проблемы аналитиков и разработчиков в области производственного планирования, но и создает основу для дальнейшего совершенствования методик интеграции аналитических решений в реальные производственные процессы, соответствуя современным тенденциям цифровизации промышленности [11,12].

4. Выводы

В ходе работы был разработан инструмент автоматизации построения систем производственного планирования, интегрированный в среду табличного редактора Univer и обеспечивающий автоматическую генерацию Python-кода на основе моделей линейного программирования. Решение опирается на библиотеку PuLP и позволяет аналитикам быстро преобразовывать формализованные задачи, представленные в виде таблиц, в исполняемый код без привлечения разработчиков, тем самым значительно сокращая время между этапами моделирования и промышленной реализации.

Представленная архитектура обладает высокой степенью расширяемости, что открывает возможности для поддержки более сложных типов задач и интеграции с другими инструментами анализа и оптимизации. Перспективным направлением является развитие интеллектуальной обработки входных данных, а также взаимодействие с промышленными ИТ-системами на уровне CI (Continuous Intelligence), что позволит инструменту стать частью комплексных решений по цифровизации производственного планирования.

Таким образом, разработанный инструмент имеет достаточно высокую практическую значимость и потенциал к дальнейшему развитию, способствуя повышению эффективности

аналитической деятельности и поддерживая современные тренды автоматизации и цифровизации в сфере управления производственными системами.

Литература

1. Alevras, Dimitris, и Manfred W. Padberg. Linear Optimization and Extensions: Problems and Solutions. Springer Science & Business Media, 2001.
2. Berg, Mark Theodoor de, и др. Computational Geometry: Algorithms and Applications. 2., rev. Ed, Springer, 2000.
3. Matoušek, Jiří, и Bernd Gärtner. Understanding and using linear programming. Springer, 2007.
4. Schrijver, Alexander. Theory of Linear and Integer Programming. Nachdr., Wiley, 2011.
5. El-Gebeily, M., и B. Yushau. «Linear System of Equations, Matrix Inversion, and Linear Programming Using MS Excel». International Journal of Mathematical Education in Science and Technology, т. 39, вып. 1, январь 2008 г., сс. 83–94.
6. Senthilnathan, Samithamby. «Solving Linear Programming Problems with the „Solver“ in MS Excel». SSRN Electronic Journal, 2014 г.
7. She, Xiangfei, и Yongchun Zheng. «An Automatic Page Code Generation Method Based on Excel Template and Poi Technology». 2020 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS), IEEE, 2020, сс. 560–64.
8. Muñoz, Lilia, и др. «Automatic Generation of ETL Processes from Conceptual Models». Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP, ACM, 2009, сс. 33–40.
9. Harris, William R., и Sumit Gulwani. «Spreadsheet Table Transformations from Examples». Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, 2011, сс. 317–28.
10. Rathod, Sunil D. «Automatic code generation with business logic by capturing attributes from user interface via XML». 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), IEEE, 2016, сс. 1480–84.
11. Sierksma, Gerard, и Diptesh Ghosh. Networks in Action: Text and Computer Exercises in Network Optimization. Springer Science+Business Media, LLC, 2010.
12. Hammer, P. L. Studies in Integer Programming. North-Holland Pub. Co. Sole distributors for the U.S.A. and Canada, Elsevier North-Holland, 1977.