Математические структуры и моделирование
2020. № 3(55). С. 72–102

УДК 510.6 : 316
DOI 10.24147/2222-8772.2020.3.72-102

ПРОГРАММНАЯ WEB-РЕАЛИЗАЦИЯ ДСМ-МЕТОДА АВТОМАТИЧЕСКОГО ПОРОЖДЕНИЯ ГИПОТЕЗ, ОСНОВАННОГО НА ИНДУКТИВНОЙ ЛОГИКЕ

Д.М. Белозеров,
магистрант, e-mail: evelend1@gmail.com
А.К. Гуц
д.ф.-м.н., профессор, e-mail: guts@omsu.ru

Омский государственный университет им. Ф.М. Достоевского, Омск, Россия

Аннотация. В статье представлено web-приложение, реализующее ДСМ-метод Финна. Приложение позволяет на основе индуктивной логики Миля извлекать новые знания из имеющейся базы знаний.

Ключевые слова: ДСМ-метод, индуктивная логика Миля, web-приложение.

В данной работе будет рассмотрен один из методов извлечения новых знаний из массивов уже существующих, известный под названием ДСМ-метода.

Автор ДСМ-метода В.К. Финн называет этот процесс knowledge discovery (обнаружение знаний) [1]. Предполагается, что знания уже содержатся в исходных данных, остаётся их только получить. В основе метода — идеи логики индуктивных рассуждений, предложенные английским логиком XIX века Д.С. Миллем. ДСМ-метод позволяет строить гипотезы на основе предыдущего опыта без каких-либо знаний о предметной области. Например, можно сделать выводы относительно свойств некоторых геометрических фигур, совсем ничего не зная об аксиомах геометрии, при этом используя полученные ранее наблюдения о других фигурах. Можно судить о свойствах сложных химических соединений, ничего не зная о химии, не производя сложное компьютерное моделирование, используя лишь индуктивные методы на массивах полученных ранее данных. Очевидно, что применений такому подходу получения знаний можно найти много.

Работы В.К. Финна положили начало исследованиям по использованию средств и языка математической логики в социологических исследованиях в России.

Первые применения разработанного В.К. Финном ДСМ-метода автоматического порождения гипотез для анализа социологических данных и порождения детерминант социального поведения представлены в двух статьях В.К. Финна и его соавторов из Института социологии РАН. Совместно с М.А. Михеенковой [2] им были разработаны методы анализа данных, применимые не только для анализа готовности к действиям, но и для анализа и прогнозирования мнений.

За рубежом начало исследованиям по применению средств и языка математической логики в социологических и политических исследованиях положили работы американского социолога Ч. Рейджина. Он изучал возможности использования булевой алгебры для формализации сравнительного анализа небольшого количества объектов. Первая публикация, описывающая метод сравнительного качественного анализа (СКА), вышла в 1987 году.

В основе логических методов, предложенных В.К. Финном и Ч. Рейджином, лежат, как уже было сказано, идеи Дж.С. Милля и его логические правила причинного вывода, согласно которым сходства, обнаруженные у объектов, обладающих определённым одинаковым свойством, являются детерминантами проявления у них этого свойства.

Для СКА идеи Дж.С. Милля (см. [3]) являются лишь одной из методологических предпосылок, в то время как с помощью ДСМ-метода все пять правил Дж.С. Милля могут быть формализованы и положены в основу рассуждений при сравнении и анализе каких-либо объектов.

СКА Рейджина и ДСМ-метод Финна «схожим образом трактуют понятие причины, а именно как сочетание условий, вызывающих определённое явление. Речь идёт о "множественной причинности подразумевающей, что для подгрупп объектов, обладающих изучаемым поведением (свойством), может быть установлено несколько моделей объяснения его наличия у них. В этой связи признаки, с помощью которых описываются объекты, рассматриваются в ходе анализа не по отдельности, а напротив, изучаются сочетания значений признаков, тем самым обеспечивается целостное описание объектов» [4].

Наша цель — создать программный инструмент для работы с ДСМ-методом [5]. Сделать его максимально простым, для того чтобы работать в системе могли люди без какой-либо специальной подготовки и особых навыков. Добавить возможность работы над задачами сразу нескольких человек. Реализуется разделение доступа между людьми для минимизации вероятности случайных ошибок и намеренных сбоев.

1. ДСМ-метод

ДСМ-метод автоматического порождения гипотез (АПГ) был предложен В.К. Финном [6]. Назван он так в честь Д.С. Милля. В.К. Финн в рассмотренных автором статьях [1, 6–8] опирается на его работы и продолжает формализацию методов Д.С. Милля. Основные идеи логики индуктивных рассуждений, предложенных Д.С. Миллем, по мнению В.К. Финна, заключаются в следующем [1]:

- индукция основывается на установлении сходства явлений изучаемой реальности;
- эти явления представлены отношением между характеристиками объектов и эффектами, которые присущи этим объектам;
- посылками индуктивного вывода является множество рассматриваемых явлений, число представителей которых больше или равно двум;

- следствием из посылок индуктивного вывода является утверждение о том, что общая часть характеристик объектов, входящих в сходные явления, есть причина некоторых общих характеристик соответствующих им эффектов (таким образом, следствие индуктивного вывода содержит новое отношение: «причина — следствие»);

- при получении индуктивного вывода относительно некоторой причины соответствующего эффекта следует установить, что не имеется причин, препятствующих наличию этого эффекта;

- достаточным основанием для индуктивного вывода является закон единообразия природы, который состоит в том, что каждый эффект наблюдаемого явления имеет свою причину.

Одной из главных идей самого ДСМ-метода АПГ является формализация взаимодействия трёх познавательных процедур — индукции, аналогии, абдукции [1].

ДСМ-метод в своей работе оперирует следующими сущностями:

- объекты;

- свойства объектов;

- структурные компоненты.

Для понимания работы будущего веб-приложения введём ещё одну сущность — задача. Задачи изолированы друг от друга. Каждая задача имеет свой набор объектов, свойств и структурных компонентов. Задачи вычисляются отдельно, поэтому далее будем считать, что речь идёт о работе с дочерними сущностями одной задачи.

У каждого объекта есть набор свойств (целевых признаков). Предполагается, что свойства не влияют друг на друга. Подмножества структурных компонентов являются предполагаемыми причинами наличия, или отсутствия свойств у объектов. Отдельный структурный компонент может либо присутствовать, либо отсутствовать у объекта. Информация о присутствии всех структурных свойств имеется в базе знаний изначально.

Перед началом работы ДСМ-метода полагаем, что уже есть некая база знаний – *база фактов*. К фактам из базы применяются строго описанные *процедуры*, основанные на индуктивной логике, дающие на каждом шаге набор формул, именуемых *гипотезами*. Их включают в *базу знаний*. Они пополняют начальную *базу фактов*:

$$\boxed{\textbf{База фактов}} \subset \boxed{\textbf{База знаний}}.$$

Процедуры могут быть двух типов, одна из которых основана на индукции, а вторая на аналогии. База знаний (фактов) содержит [6]:

- O — множество объектов;
- P — множество свойств объектов (целевых признаков);
- C — множество элементов структуры (возможных причин свойств);
- V — множество значений свойств. В нашем случае это {1, -1, 0, -};
- функцию P: O => 2^C, которая определяет наличие у объектов подмножества целевых признаков;
- функцию F: O*P => V, которая определяет начальное значение свойств объектов.

Значения функции F из области V можно описать следующим образом [6]:

- F(o, p) = 1 — объект o обладает свойством p;
- F(o, p) = -1 — объект o не обладает свойством p;
- F(o, p) = 0 — есть противоречивые аргументы относительно наличия свойства p у объекта o;
- F(o, p) = - — заранее не известно о наличии свойства p у объекта o.

Задача ДСМ-метода состоит в том, чтобы доопределить все значения функции F, содержащей значение {-} до одного из значений {1, -1, 0}. Происходит это путём формирования гипотез:

- о связи подмножества структурных признаков со значением свойства. Это гипотезы о возможных причинах свойств;
- о значении свойства для объектов, изначально имевших значение {-} (путём применения гипотез о возможных причинах свойств).

Гипотезы о возможных причинах свойств обозначим функцией H: C*P => V. Значения этой функции можно описать следующим образом [2]:

- H(c, p) = 1 — подмножество структурных признаков c является возможной причиной наличия свойства p;
- H(c, p) = -1 — подмножество структурных признаков c является возможной причиной отсутствия свойства p;
- H(c, p) = 0 — есть аргументы как за, так и против того, что подмножество структурных признаков c является причиной свойства p;
- H(c, p) = - — нет информации о связи подмножества структурных признаков c со свойством p.

Итак, ДСМ-метод заключается в последовательном применении познавательных процедур [1]:

- индукции — правила вывода первого рода для получения гипотез о возможных причинах свойств (функция H);
- аналогии — правила вывода второго рода для получения гипотез о наличии свойств у объектов (доопределение функции F);
- абдукции — для проверки каузальной полноты.

Правила вывода первого и второго рода последовательно применяются до тех пор, пока удаётся сгенерировать ещё хотя бы одну гипотезу и пока имеются недоопределённые объекты. Считается что чем больше шагов потребовалось для генерирования очередной гипотезы, тем меньше её достоверность, так как она опирается на результаты других гипотез.

Абдукция применяется только после того, как завершено применение правил первого и второго рода.

Правило первого рода — это некоторая функция, использующая базу знаний для получения функции H. Определим правило первого рода как H = PIR1(F). Далее опишем алгоритм работы правил первого рода.

Можно поделить все объекты o в базе на:

- положительные примеры для свойства p относительно матрицы F, если F(o, p) = 1;
- отрицательные примеры для свойства p относительно матрицы F, если F(o, p) = -1;
- противоречивые примеры для свойства p относительно матрицы F, если F(o, p) = 0.

Объекты, для которых справедливо: F(o, p) = - не рассматриваются, так как они не участвуют в формировании функции H.

Для каждого набора структурных признаков необходимо найти множество положительных, отрицательных и противоречивых примеров соответственно. Введём обозначения для этого:

- M+(F, c, p) — набор структурных признаков c вкладывается хотя бы в один объект, являющийся положительным примером относительно F для свойства p;
- M-(F, c, p) — набор структурных признаков c вкладывается хотя бы в один объект, являющийся отрицательным примером относительно F для свойства p;
- M0(F, c, p) — набор структурных признаков c вкладывается хотя бы в один объект, являющийся противоречивым примером относительно F для свойства p.

Теперь можно определить функцию H(c, p) следующими значениями [2]:

- 1, если $M+(F, c, p)\wedge\neg M-(F, c, p)\wedge\neg M0(F, c, p)$,
- -1, если $M-(F, c, p)\wedge\neg M+(F, c, p)\wedge\neg M0(F, c, p)$,
- 0, если $(M+(F, c, p)\wedge M-(F, c, p))\vee M0(F, c, p)$,
- -, если $\neg M+(F, c, p)\wedge\neg M-(F, c, p)\wedge\neg M0(F, c, p)$.

На этом действие индукции завершено, гипотезы сформированы.

Правила вывода второго рода преобразуют функцию F, доопределяя некоторые её значения. Обозначим новую функцию F1 как доопределённую в результате применения аналогии функцию F. Тогда функция F1(o, p) будет принимать значения исходной функции F, если F(o, p) = {1, -1, 0}. Остальные значения определяются применением правил второго рода при использовании матрицы H: F1 = PIR2(F, H).

Аналогично правилам первого рода введём новые обозначения:

- K+(H, o, p) — в объект o вкладывается хотя бы одна гипотеза из H о наличии свойства p,
- K-(H, o, p) — в объект o вкладывается хотя бы одна гипотеза из H об отсутствии свойства p,
- K0(H, o, p) — в объект o вкладывается хотя бы одна противоречивая гипотеза из H.

Тогда доопределим функцию F1 следующими значениями:

- 1, если $K+(H, o,p)\wedge\neg K-(H, o,p)\wedge\neg K0(H, o,p)$,
- -1, если $K-(H, o,p)\wedge\neg K+(H, o,p)\wedge\neg K0(H, o,p)$,
- 0, если $(K+(H, o,p)\wedge K-(H, o,p))\vee K0(H, o,p)$,
- -, если $\neg K+(H, o,p)\wedge\neg K-(H, o,p)\wedge\neg K0(H, o,p)$.

На этом заканчивается применение правил вывода второго рода. После того, как правила вывода первого и второго рода попеременно применены максимальное количество раз, проведём проверку каузальной полноты.

С этой целью для всех объектов в функции F(o, p), значения для которых были известны изначально, необходимо проверить, что имеется хотя бы одна гипотеза из H(c, p), которая объясняла бы это значение [6]. Если объяснения начальных значений найдены, то проверка каузальной полноты считается пройденной и результаты ДСМ-метода можно применять. В противном случае рекомендуется расширить базу фактов, пересмотреть набор структурных признаков или способ кодирования структурных признаков.

2. Реализация web-приложения

В этой главе будут описаны основные особенности реализации, используемые подходы и инструменты для разработки web-приложения.

2.1. Используемые инструменты для разработки

В качестве среды разработки используется Microsoft Visual Studio 2017 RC [9]. Версия release candidate используется для тестирования среды разработки и распространяется без ограничений. Преимуществом среды разработки Visual Studio является её широкое распространение, и, следовательно, большее количество справочного материала, создаваемого в том числе сообществом пользователей. Также преимуществом для авторов статьи является хорошая осведомлённость о возможностях и особенностях Visual Studio.

Используется система контроля версий TFS [10]. Она позволяет создавать версии исходного кода web-приложения, смотреть изменения, вносимые в код, повышает удобство дальнейшей работы с кодом. Также код теперь хранится не только на устройствах авторов, но и на серверах Microsoft, что уменьшает вероятность потерять результаты работы. Адрес репозитория: https://evelent.visualstudio.com/JSMethod/. Репозиторий закрытый, для получения к нему доступа необходимо связаться с авторами. Visual Studio по умолчанию имеет интеграцию с TFS, так что делать коммиты можно прямо из среды разработки.

В своей web-разработке авторы старались использовать наиболее современные технологии, даже в ущерб плохой их поддержке старыми браузерами, например, IE9 (поддержка которого закончилась в 2016 году [11]). По этой причине рекомендуется для работы с системой использовать один из современных браузеров последней версии. Используемые технологии будут описаны ниже.

Для создания web-приложения был выбран фреймворк ASP.NET MVC [12] на движке Razor [13], файлы представлений в котором не являются статичными, а генерируют ответ каждый раз в зависимости от состояния моделей.

С целью упрощения и ускорения создания оформления на web-приложении был применён HTML, CSS и Javascript фреймворк Bootstrap [14]. Не стояло задачи делать какой-либо уникальный дизайн, а оформление от Bootstrap по умолчанию выглядит очень хорошо. Фреймворк также позволил сократить трудозатраты на создание адаптивной вёрстки, при которой сайтом удобно пользоваться на устройствах с различной шириной экрана, и на создание простого меню для сайта.

Естественно, использовалась JavaScript библиотека jQuery [15]. Она помогает легко взаимодействовать с деревом элементов браузера, изменять атрибуты элементов, предоставляет удобный интерфейс для AJAX [16] запросов и требуется для работы некоторых других компонентов, используемых в разработке.

Создание таблиц облегчил клиентский фреймворк Kendo UI [17]. Встроенный компонент Kendo Grid [18] реализует стандартное поведение таблиц, имеет поддержку серверной пагинации, гибкие настройки и достаточно полную документацию всего функционала на сайте разработчика системы.

Экспорт в Excel производится с помощью интеграции фреймворка Kendo UI и библиотеки JSZip [19]. Библиотека способна сформировать Excel файл сразу на стороне клиента, используя данные из таблиц Kendo Grid. Такой подход уменьшает нагрузку на сервер, снимает необходимость реализовывать экспорт самостоятельно.

2.2. Особенности реализации

Для удобного подключения всех используемых JavaScript и CSS файлов применяется BundleConfig [20]. Он позволяет задать соответствие между конкретными файлами и абстрактными сущностями, которые можно подключить на страницу. Так как скриптов и файлов стилей в проекте используется немного, и они почти всегда все нужны на всех страницах, то было создано всего две сущности:

«/Content/js» и «/Content/css», пример добавления сопоставления показан ниже:

```
bundles.Add(new ScriptBundle("~/Content/js" ).Include( // добавление сущности
    "~/Content/js/jquery-{version}.js" , // привязка файла к сущности
```

Также стоит заметить, что BundleConfig позволяет использовать маски в именах файлов, например, «version». Таким образом, после замены библиотеки на новую версию не надо будет что-либо менять в коде.

В основной модели данных ДСМ-метода используются следующие сущности:

- задача (класс Problem);
- объект (класс Subject);
- свойство (класс Property);
- элемент структуры (класс Composition);
- значение свойства (класс ValueOfProperty);
- значение элемента структуры (класс ValueOfComposition).

Отдельные классы для значений свойств и структурных признаков созданы для удобства хранения их в базе данных. К тому же заранее не известны количество и состав свойств. Собственно, свойств, описывающих состояние самого класса, мало. Для задач, свойств и объектов это свойство: имя (Name), для классов, отвечающих за хранение значения — значение (Value).

Отношения между классами показаны ниже (рис. 1).

Работа напрямую с базой данных не ведётся. Используется подход Code First и Entity фреймворк [21] в качестве инструмента. Преимущества подхода Code First:

- таблицы в базе данных генерируются автоматически;

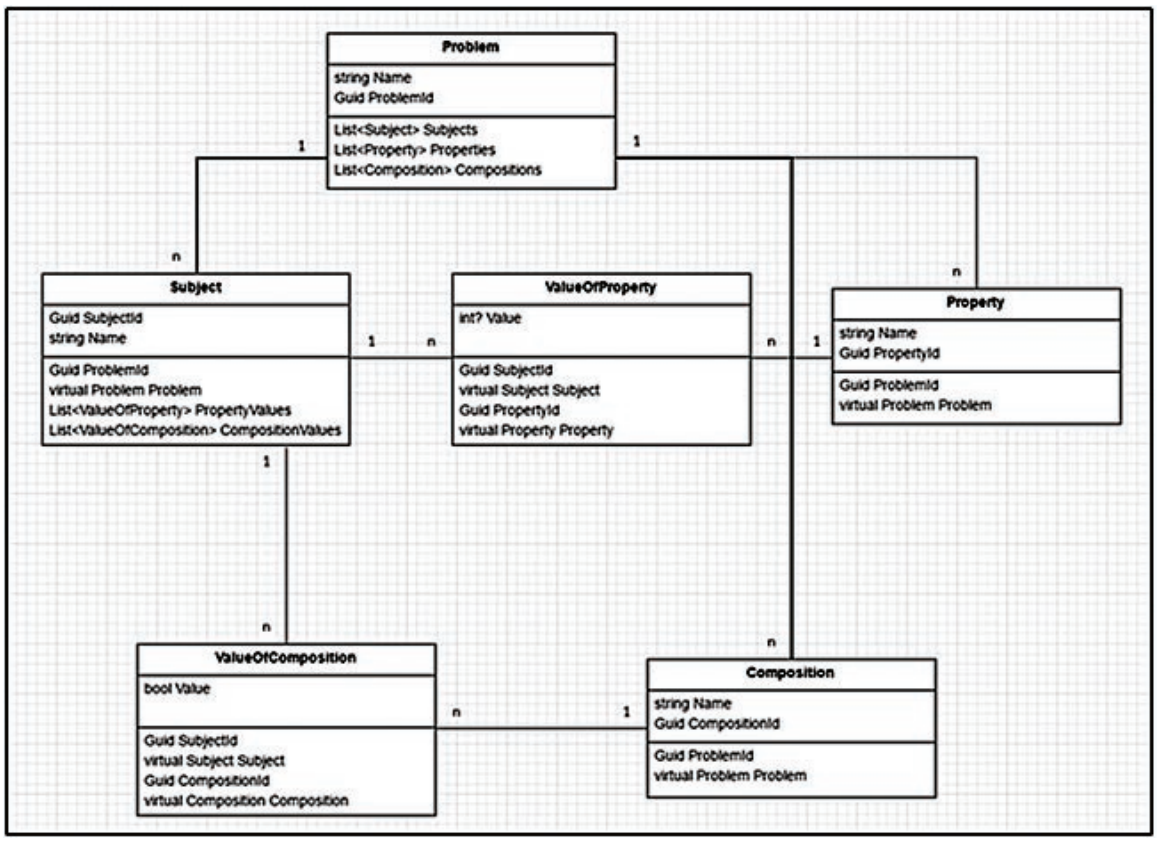


Рис. 1. Диаграмма классов сущностей ДСМ-метода.

- сопоставления между базой данных и программными объектами не нужно описывать отдельно;
- запросы к базе на языке sql генерируются автоматически, прозрачно для пользователя;
- простое изменение схемы данных, изменения для базы генерируются автоматически;
- установка среды разработки происходит автоматически с помощью системы миграций.

Таким образом, просто описав систему классов, показанную выше, можно сразу переходить к работе с ДСМ-методом и не заботиться о том, как данные будут храниться в базе.

Данные заносятся в систему с помощью контроллера DataInputController. Возможность вводить данные есть не у всех пользователей, а только у тех, кто принадлежит к группе «Операторы». Реализуется это при помощи AspNet Identity [22] библиотеки, а в классе описывается добавлением атрибута:
[Authorize(Roles = Const.OperatorRole)]

Где атрибут Authorize указывает, что пользователь обязательно должен быть авторизован для работы с контроллером ввода данных и иметь роль «Оператор», иначе ни один запрос, даже GET к API получения данных, не пройдёт [23]. Принадлежность пользователя к ролям настраивается администратором системы через web-интерфейс.

Контроллер DataInputController обслуживает запросы с трёх страниц: Problems, Problem, Subjects. Страница Problems в табличном виде выводит все созданные ранее задачи, позволяет добавлять новые задачи, удалять и переименовывать уже созданные. Страница Problem предназначена для изменения описания задачи. А именно: состав свойств и структурных признаков. Редактирование доступно также в табличном виде, как и для задач. Страница Subjects нужна для редактирования набора объектов задачи. Таблица для редактирования объектов более сложная, т. к. содержит переменное количество колонок и позволяет делать отправку сразу нескольких объектов на сервер в одном запросе.

Опишу более подробно функционирование страницы Subjects. Таблица на стороне клиента строится с плагином Kendo Grid. Выглядит это следующим образом:

```
    jQuery(document).ready(function () {
//Код выполняется после загрузки документа
    $("#subjectsTable" ).kendoGrid({ //Указывается таблица
    dataSource: { //Описываются источники данных
    type: "json" ,
    transport: {
    create: {
    url: "@Url.Action("Subjects_Create" , new { problemId =
Model.Problem.ProblemId })" , //Привязка действия для создания элемента
//Тут же указывается дополнительный параметр problemId [20]
    dataType: "json" , //Описание типа данных
    type: "POST"//Тип запроса POST [21]
    },
    read: {
    url: "@Url.Action("Subjects_Read" , new { problemId =
Model.Problem.ProblemId })" , //Привязка действия для чтения элементов
    dataType: "json" ,
    type: "POST"
    },
    update: {
    url: "@Url.Action("Subjects_Update" , new { problemId =
Model.Problem.ProblemId })" , //Привязка действия для обновления элементов
    dataType: "json" ,
    type: "POST"
    },
    destroy: {
    url: "@Url.Action("Subjects_Destroy" , new { problemId =
Model.Problem.ProblemId })" , //Привязка действия для удаления элемента
```

```
dataType: "json" ,
type: "POST"
}
},
pageSize: 10, //Размер страниц при пагинации
schema: {
model: { //Описание схемы данных модели
id: "SubjectId" ,
fields: @Html.Raw(Model.Fields) //Поле Fields модели //данных возвращает
описание столбцов таблицы
},
total: "total" ,
data: "data"
},
serverPaging: true, //Включение серверной пагинации
batch: true //Включение режима отправки многих элементов
},
height: 550, //Высота таблицы
toolbar: ["create" , "save" , "cancel"], //Кнопки в меню таблицы
pageable: { //Настройка пагинатора
refresh: true, //Показывать кнопку обновления
pageSizes: true, //Показывать кнопку выбора размера страницы
buttonCount: 5,
pageSizes: [10, 20, 50] //Возможные размеры страницы
},
columns: @Html.Raw(Model.GetColumns(150, 50, 110, true)),
//GetColumns возвращает описание колонок таблицы (с именами)
editable: true //Включение возможности редактирования элементов
});
});
```

Как видно, код для описания таблицы действительно очень прост. Теперь стоит показать, как выглядит метод получения объектов на стороне сервера. Стоит заметить, что все методы, в которых есть работа с базой данных, выполняются на сервере асинхронно (ключевое слово async [26]), и в момент запроса к базе метод ставится на паузу, поток освобождается для обработки других запросов. Это позволяет значительно повысить пропускную способность сервера.

```
public async Task<ActionResult> Subjects_Read(Guid problemId, int? skip, int?
take) //Входные параметры автоматически извлекаются из запроса
{
var allItemsQuery = JsmContext.Subjects.Where(i => i.ProblemId ==
problemId);
//Создание запроса для получения всех объектов из базы, объекты в этот //мо-
мент ещё не запрашиваются
var query = allItemsQuery;
query = query.OrderBy(p => p.Name); //Указываем что необходима //сорти-
ровка
```

```
    if (skip.HasValue) //Далее пропускаем необходимое количество //элементов и
получаем необходимое количество элементов. Так реализуется //серверная пагинация
    query = query.Skip(skip.Value);
    if (take.HasValue)
    query = query.Take(take.Value);
    var items = await query.Include(x => x.PropertyValues).Include(x =>
x.CompositionValues).ToListAsync();
//Метод Include указывает что необходимо так же извлечь объекты, //отвечающие за хранение значений свойств и структурных компонентов [27]. //Await
позволяет поставить выполнение метода на паузу и освободить поток, до //тех пор, пока не будет выполнен запрос к базе
    SubjectsViewVodel subjectsModel = await SubjectsViewVodel.FromDb(problemId,
JsmContext);
//Получение объектов из базы
    var models = items.Select(dm =>
    subjectsModel.FromDataModel(dm)).ToList();
//Формирование модели представления для отправки клиенту
    var response = new //Формирование ответа
    {
    total = await allItemsQuery.CountAsync(), //Количество всех //элементов
    data = models
    };
//Отправка ответа клиенту в формате Json [28]
    return Json(response, JsonRequestBehavior.AllowGet);
    }
```

Методы добавления, изменения, удаления объектов выглядят аналогично. Разница только в логике обращения к базе, но она проста во всех случаях, рассмотрения одного этого метода достаточно.

Теперь, когда способ ввода данных в систему определён, перейдём к выполнению вычислений. Для выполнения вычислений создан контроллер CalculatingController. И две страницы, которые этот контроллер обслуживает: Problems и Result. Доступ к работе с этим разделом сайта имеют все зарегистрированные пользователи. Открыть доступ для всех можно по причине того, что из этого контроллера пользователь никак не может изменить данные. Всё, что можно сделать — пользоваться результатами вычислений ДСМ-метода. Проведение вычислений так же не затрагивает начальное состояние моделей.

На странице Problems отображаются все задачи. При клике на имя задачи происходит переход к представлению Result с одновременной передачей идентификатора задачи в это представление.

Вычисления начинаются в момент формирования ответа на запрос представления Result. Когда представление отображается пользователю, все вычисления уже проведены и диагностическая информация уже собрана. Также страница результатов содержит таблицу с объектами. В отличие от страницы с объектами в контроллере DataInputController тут не поддерживается редактирование объектов, пагинация производится на стороне клиента (так как все данные

уже получены), и те свойства, которые не были определены раньше, теперь имеют определённые значения. В этом и состоит задача web-приложения — доопределить свойства объектов.

В таблице также имеется возможность экспорта результатов вычислений в Excel.

Помимо таблицы объектов на странице Results содержится раскрываемая область с диагностическими данными о ходе выполнения ДСМ-метода. Вывод этих данных был добавлен, так как только по одному состоянию объектов не всегда можно судить о правильности выполнения метода. Также иногда полезно посмотреть, какие гипотезы были сгенерированы на определённом шаге, какие свойства, в какой последовательности доопределены. В лог выводится следующая информация:

- начало и окончание расчёта определённого свойства;
- количество объектов, для которых не определено значение свойства на каждом шаге;
- возможные причины наличия / отсутствия структурных признаков и противоречивые гипотезы;
- успешно ли пройдена проверка каузальной полноты;
- какие объекты и почему не удовлетворяют проверке каузальной полноты;
- какие свойства доопределены у каких объектов.

Рассмотрим обработчик запроса страницы Result.

```
public async Task<ActionResult> Result(Guid problemId)
{ //Как и другие обработчики выполняется асинхронно
//Сначала идет получение задачи из базы, включая все связанные значения
Problem problem = await JsmContext.Problems
.Include(i => i.Properties).Include(i => i.Compositions).Include(i => i.Subjects)
.Include(i => i.Subjects.Select(s => s.PropertyValues)).Include(i => i.Subjects.Select(s => s.CompositionValues))
.SingleOrDefaultAsync(i => i.ProblemId == problemId);
var calc = new JsmCalculator(problem);
calc.Calculate(); //Создание объекта «калькулятора» и //произведение вычислений (подробнее будет рассмотрено ниже)
string log = calc.Log.Replace(Environment.NewLine, @"<br />" );
//Получение логов вычислений
SubjectsViewVodel model = await SubjectsViewVodel.FromDb(problemId, JsmContext);
model.Log = log;
//Формирование массива объектов для таблицы результатов
//Объект problem уже содержит конечное состояние задачи
```

```
    model.Subjects = problem.Subjects.Select(dm => model.FromDataModel(dm)).ToList();
//Возврат результата
    return View(model);
    }
```

Теперь более подробно рассмотрим, как работает класс, отвечающий за вычисления ДСМ-метода.

Основная функция приведена ниже. Тут объект Problem — свойство класса JsmCalculator. Каждое целевое свойство рассчитывается отдельно. Расчёт следующего свойства начинается после прохождения всех шагов вычисления предыдущего, так как в ДСМ-методе нет зависимости между разными свойствами (только зависимость между структурными признаками и свойством).

```
    public void Calculate()
    {
    WriteLog($"Начало вычислений для задачи с именем: '{Problem.Name}'." );
    foreach (var property in Problem.Properties)
    { //Проход по всем свойствам
    int pastNullCount = Int32.MaxValue;
    int nullCount = ToCountNullPropValues(property);
    int stepCount = 0;
    WriteLog($"Свойство: '{property.Name}'. Неопределённых объектов: {nullCount}." , true);
    var plusExamples = new List<CalcExample>();
    var minusExamples = new List<CalcExample>();
    var contradictoryExamples = new List<CalcExample>();
    while (nullCount != 0 && nullCount != pastNullCount)
//Расчёт ведётся пока не доопределены значения свойства для всех объектов, и
//пока удаётся найти хотя бы одно значение свойства за один шаг
    {
    stepCount++;
    WriteLog($"Шаг № {stepCount}." , true);
    pastNullCount = nullCount;
    CalculateStep(property, ref plusExamples, ref minusExamples, ref contradictoryExamples); //Функция для вычисления шага ДСМ-метода
    nullCount = ToCountNullPropValues(property);
    WriteLog($"Осталось неопределённых объектов: {nullCount}." );
    }
    bool causalCompleteness = CheckConditionOfCausalCompleteness(property, plusExamples, minusExamples);
//Проверка каузальной полноты
    if (causalCompleteness)
    WriteLog($"Проверка каузальной полноты пройдена успешно.");
    }
    WriteLog($"Вычисления завершены." , true);
    }
```

Далее рассмотрим метод CalculateStep. Расчёт шага подразумевает последовательное выполнение правил первого рода (процедура индукции) и правил второго рода (процедура аналогии). Из метода CalculateStep происходит вызов соответствующих методов, поэтому листинг кода приводиться не будет. GetPossibleReasons отвечает за выполнение процедуры индукции, DefineObjects — процедуры аналогии.

Процедура аналогии также поделена на методы с высоким уровнем абстракции. Оттуда происходит поочерёдный вызов следующих методов:

1. FindSubsetsOfCompositions. Поиск возможных наборов структурных признаков для гипотез.

2. CheckSubsets. Для проверки принадлежности каждого из наборов структурных признаков к положительным, отрицательным и противоречивым.

3. RemoveNestedContradictoryExamples. Для удаления вложенных противоречивых примеров из списка гипотез.

Метод FindSubsetsOfCompositions представляется важным, поэтому рассмотрим его с комментариями. Задачей этого метода является поиск всех наборов структурных признаков. Для этого необходимо рассмотреть все возможные их комбинации, которых два в степени их количества.

```
void FindSubsetsOfCompositions(Property property, ref List<CalcExample> plusExamples, ref List<CalcExample> minusExamples, ref List<CalcExample> contradictoryExamples)
{
int combinationsCount = (int)Math.Pow(2, Problem.Compositions.Count);
//Просматриваются все возможные комбинации
for (int i = 1; i < combinationsCount; i++)
{
List<Composition> includedComposition = new List<Composition>(Problem.Compositions.Count);
var includeCombination = Convert.ToString(i, 2);
//Число — это «код» возможных комбинаций, где единицами обозначается
//включение структурного признака
for(int j = includeCombination.Count() - 1; j >= 0; j–)
{
if (includeCombination[j] == '1')
{
int index = Problem.Compositions.Count - includeCombination.Length + j;
includedComposition.Add(Problem.Compositions[index]);
}
}
//На данном этапе в переменной includeCombination все включённые //комбинации
List<Subject> plusSubjects = new List<Subject>();
List<Subject> minusSubjects = new List<Subject>();
```

```
    List<Subject> zeroSubjects = new List<Subject>();
    foreach (var subject in Problem.Subjects)
    {
//Проверяем все объекты на наличие текущего набора структурных признаков
    bool includedAllComposition = true;
    foreach (var composition in includedComposition)
    {
    if (!subject.CompositionValues.Single(cv => cv.CompositionId ==
composition.CompositionId).Value)
    includedAllComposition = false;
    }
    if (includedAllComposition)
    {
    var value = subject.PropertyValues.Single(pv => pv.PropertyId ==
property.PropertyId).Value;
//Если весь набор включён, то добавляем этот объект как пример
    if (value == 1)
    plusSubjects.Add(subject);
    else if (value == -1)
    minusSubjects.Add(subject);
    else if (value == 0)
    zeroSubjects.Add(subject);
    }
    }
//Если объектов больше определённого количества, то добавляем это набор как
//возможную гипотезу
    if (plusSubjects.Count >= _MinExamplesCount)
    AddExample(plusExamples, includedComposition, plusSubjects);
    if (minusSubjects.Count >= _MinExamplesCount)
    AddExample(minusExamples, includedComposition, minusSubjects);
    if (zeroSubjects.Count >= _MinExamplesCount)
    AddExample(contradictoryExamples, includedComposition, zeroSubjects);
    }
    }
```

Метод FindSubsetsOfCompositions имеет самую большую вычислительную сложность в вычислениях ДСМ-метода (O(s*2∧n), где n — количество структурных признаков, s — количество объектов).

Следующий метод — CheckSubsets. Теперь, когда возможные гипотезы найдены, необходимо проверить, удовлетворяют ли они всем объектам или некоторые объекты будут служить контрпримером для них.

```
    void CheckSubsets(Property property, ref List<CalcExample>
plusExamples, ref List<CalcExample> minusExamples, ref List<CalcExample>
contradictoryExamples)
    {
//Складываем все примеры в один массив
    List<CalcExample> allExamples =
    plusExamples.Union(minusExamples).Union(contradictoryExamples).ToList();
```

```
    plusExamples.Clear();
    minusExamples.Clear();
    contradictoryExamples.Clear();
//Остальные массивы очищаем (они будут перезаполнены после перепроверки)
    foreach (var example in allExamples)
    {
    int plusCount = 0;
    int minusCount = 0;
    int zeroCount = 0;
    foreach (var subject in Problem.Subjects)
    {
    var subjectsCompositionValues = subject.CompositionValues.Where(cv =>
example.Compositions.Any(c => c.CompositionId == cv.CompositionId));
    if (subjectsCompositionValues.All(scv => scv.Value == true))
    {
    var value = subject.PropertyValues.Single(pv => pv.PropertyId ==
property.PropertyId).Value;
    if (value == 1)
    plusCount++;
    else if (value == -1)
    minusCount++;
    else if (value == 0)
    zeroCount++;
//Производится подсчёт свойств с положительными, отрицательными и //нулевыми значениями
    }
    }
    if (plusCount >= _MinExamplesCount && minusCount == 0 && zeroCount ==
0)
//В соответствии с логикой ДСМ-метода, если у набора признаков только //положительные примере, то он становится положительной гипотезой
    plusExamples.Add(example);
    else if (minusCount >= _MinExamplesCount && plusCount == 0 && zeroCount
== 0)
    minusExamples.Add(example);
//Если только отрицательные — отрицательной
    else if (zeroCount > 0 || (plusCount > 0 && minusCount > 0))
    contradictoryExamples.Add(example);
//Если как положительные, так и отрицательные — противоречивой
    }
    }
```

На выходе из этого метода мы получаем набор готовых гипотез. Однако их необходимо отфильтровать. Рассмотрим пример, допустим у нас есть положительный набор структурных признаков: {c1, c2}, отрицательный набор: {c1, c3} и противоречивый: {c1}. Тут c1 — включён как в положительный набор, так и в отрицательный, очевидно, что гипотеза {{c1} — следует противоречие} не будет представлять ценность при определении свойств

объектов, а даже будет «мешать» другим, более полным гипотезам. Поэтому такие гипотезы следует удалить из набора. За это и отвечает метод RemoveNestedContradictoryExamples, который достаточно прост в реализации и подробно рассматриваться не будет.

Напоминаю, что на выходе метода GetPossibleReasons, который мы рассмотрели, будут массивы с готовыми гипотезами. Гипотезы сами по себе уже являются новым знанием, но их ещё можно применить для определения свойств объектов, которые были не известны до этого. Рассмотрим метод DefineObjects.

```
    void DefineObjects(Property property, List<CalcExample> plusExamples,
List<CalcExample> minusExamples, List<CalcExample> contradictoryExamples)
    {
//Просматриваются все объекты
    foreach (var subject in Problem.Subjects)
    {
    var propertyValue = subject.PropertyValues.Single(pv => pv.PropertyId ==
property.PropertyId);
    if (propertyValue.Value != null)
    continue; //Если свойство уже задано, происходит переход на //следующую
итерацию цикла
    bool satisfiesPlusExample = plusExamples.Any(ex =>
CheckSubjectIsSatisfiesExample(subject, ex));
//Проверка сколько гипотез разного типа можно применить к объекту
    bool satisfiesMinusExample = minusExamples.Any(ex =>
CheckSubjectIsSatisfiesExample(subject, ex));
    bool satisfiescontradictoryExample = contradictoryExamples.Any(ex =>
CheckSubjectIsSatisfiesExample(subject, ex));
    bool define = false;
    if (satisfiesPlusExample && !satisfiesMinusExample &&
!satisfiescontradictoryExample)
    {
//В соответствии с логикой ДСМ-метода применяем гипотезы
//И задаем значение свойства равным 1, если объект удовлетворяет только //по-
ложительным гипотезам
    propertyValue.Value = 1;
    define = true;
    }
    else if (!satisfiesPlusExample && satisfiesMinusExample &&
!satisfiescontradictoryExample)
    {
//Равным минус единице, если только отрицательным
    propertyValue.Value = -1;
    define = true;
    }
    else if (satisfiescontradictoryExample || (satisfiesPlusExample &&
satisfiesMinusExample))
    {
    propertyValue.Value = 0;
```

```
define = true;
}
//И нулю, если существуют факты как положительные, так и отрицательные
if (define)
WriteLog($"Доопределено свойство '{property.Name}' объекта '{subject.Name}' значением '{propertyValue.Value}'");
}
}
```

Теперь, после выполнения нескольких шагов индукции и аналогии необходимо выполнить проверку каузальной полноты. Проверка пройдена, если все положительные и отрицательные примеры свойств можно объяснить с помощью полученных гипотез, рассмотрим метод.

```
bool CheckConditionOfCausalCompleteness(Property property, List<CalcExample> plusExamples, List<CalcExample> minusExamples)
{
bool success = true;
foreach(var subject in Problem.Subjects)
{
//Проверяем все объекты
var propertyValue = subject.PropertyValues.Single(pv => pv.PropertyId == property.PropertyId);
if ((propertyValue.Value == 1 && !plusExamples.Any(ex => CheckSubjectIsSatisfiesExample(subject, ex)))
|| (propertyValue.Value == -1 && !minusExamples.Any(ex => CheckSubjectIsSatisfiesExample(subject, ex))))
{
//Если какой-либо из них не может быть обоснован, то проверка не пройдена
success = false;
WriteLog($"Объект '{subject.Name}' по свойству '{property.Name}' не удовлетворяет условию каузальной полноты.");
//В лог сразу записывается информация об объектах, ставших причиной //невыполнения проверки
}
}
return success;
}
```

После прохождения проверки каузальной полноты работу ДСМ-метода можно считать завершённой. Если проверка каузальной полноты не была выполнена успешно, то использовать его результаты не рекомендуется, вероятнее всего, необходимо дополнить начальную базу фактов большим количеством примеров.

Таким образом, в данной главе были рассмотрены наиболее важные участки кода по реализации ДСМ-метода.

3. Результаты

Результатом разработок стало web-приложение, позволяющее работать с ДСМ-методом без каких-либо специальных навыков. В этом параграфе web-приложение рассматривается со стороны web-интерфейса, показано, как пользователи могут с ним работать. При первом входе пользователь попадает на главную страницу. Так как он пока не зарегистрирован, для него отображается неполное верхнее меню и нет доступа к основным функциям (рис. 2).

На различных устройствах web-приложение может выглядеть по-разному из-за использования подхода адаптивной верстки [29].

Ниже продемонстрирован вид главной страницы на устройствах с узким экраном (рис. 3), можно заметить, что меню при этом также принимает иной вид (рис. 4).

Далее пользователь может пройти регистрацию, указав свою электронную почту и пароль. Пароль отправляется на сервер в зашифрованном виде, и в базе данных хранится только его хэш. За эти операции отвечает фреймворк asp identity. После регистрации пользователю доступна привычная форма для входа на сайт (рис. 5).

У пользователей могут быть разные права доступа. Принадлежность к группам доступа настраивает администратор системы (рис. 6).

Из служебных страниц также имеется страница управления пользователем, откуда он может перейти на страницу смены пароля или удалить собственный аккаунт (рис. 7).

Теперь перейдём непосредственно к работе с ДСМ-методом. О том, как функционирует страница добавления задач, было сказано в предыдущем параграфе, здесь обратим внимание на внешний вид и удобство использования. Задачи можно редактировать при помощи таблицы (рис. 8). Тут доступны все CRUD [30] операции. При удалении задачи будут удалены все связанные с ней записи из базы данных. При клике на название задачи происходит переход на страницу настроек её свойств. Ссылка «объекты» ведет сразу к странице настроек объектов.

На всех скриншотах будет показана работа с известным примером ДСМ-метода из вики: можно ли описать окружность вокруг прямоугольника. Вывод будет сделан на основе таких структурных признаков, как наличие оси симметрии, наличие прямого угла и других. Следующий скриншот показывает страницу после перехода по названию задачи (рис. 9).

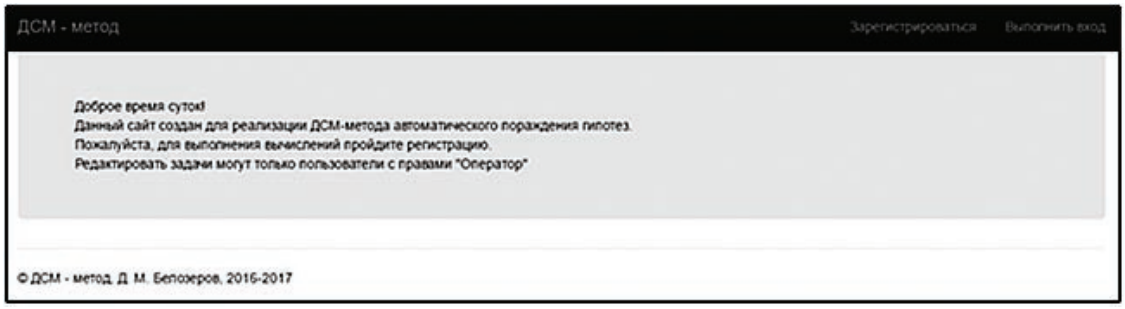


Рис. 2. Домашняя страница

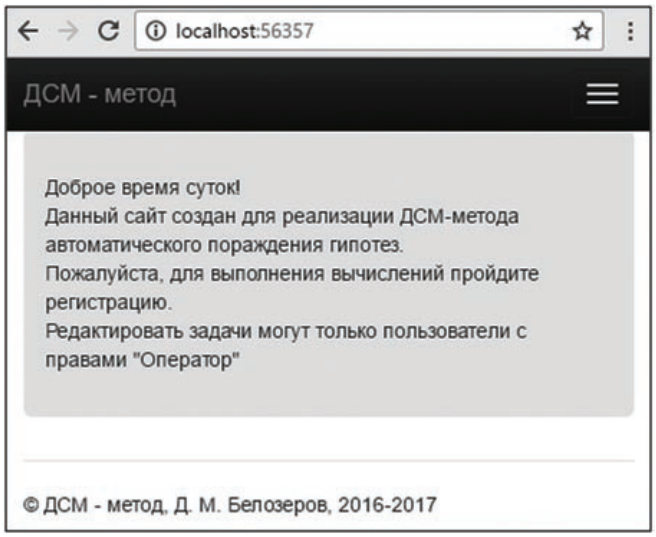


Рис. 3. Вид главной страницы на устройствах с узким экраном

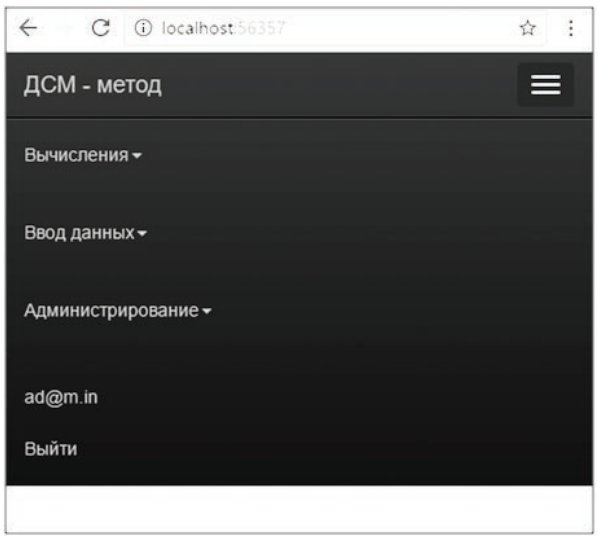


Рис. 4. Вид меню на устройствах с узким экраном

Тут отображается название задачи (изменять его можно с предыдущей

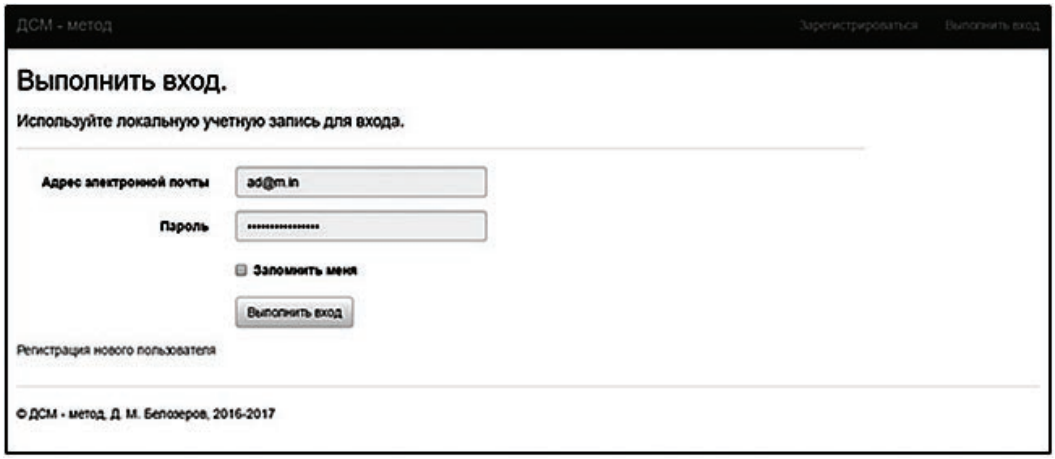


Рис. 5. Форма авторизации пользователя

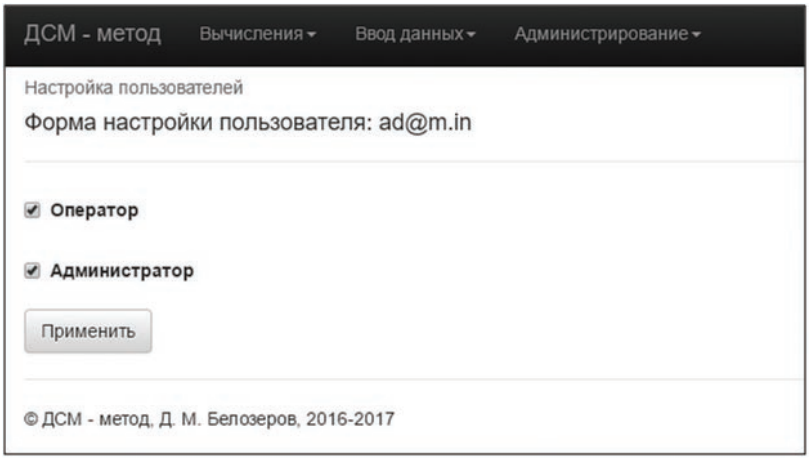


Рис. 6. Настройка доступа

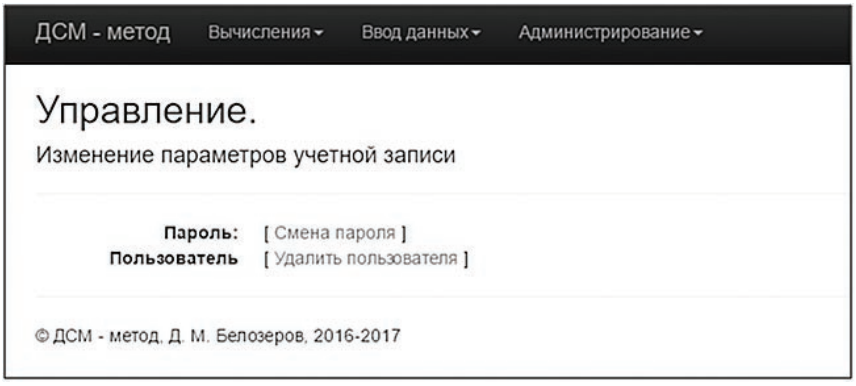


Рис. 7. Страница управление профилем

страницы), ссылки на возврат ко всем задачам и к странице описания объектов. Целевые свойства не обязательно должны иметь уникальное имя, так как уникальность обеспечивается уникальным идентификатором [31], но это

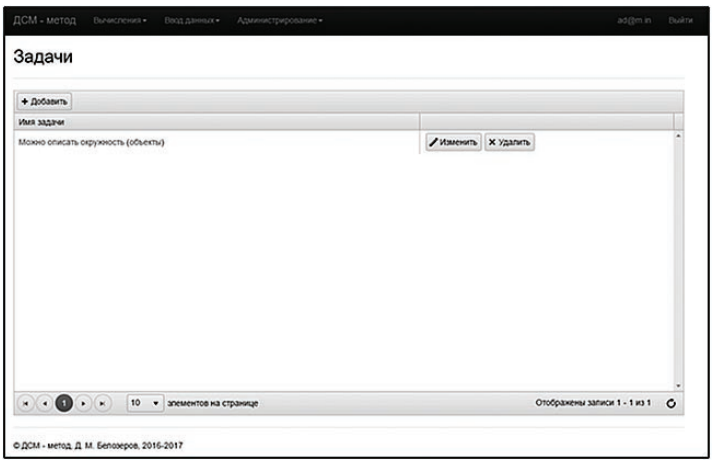
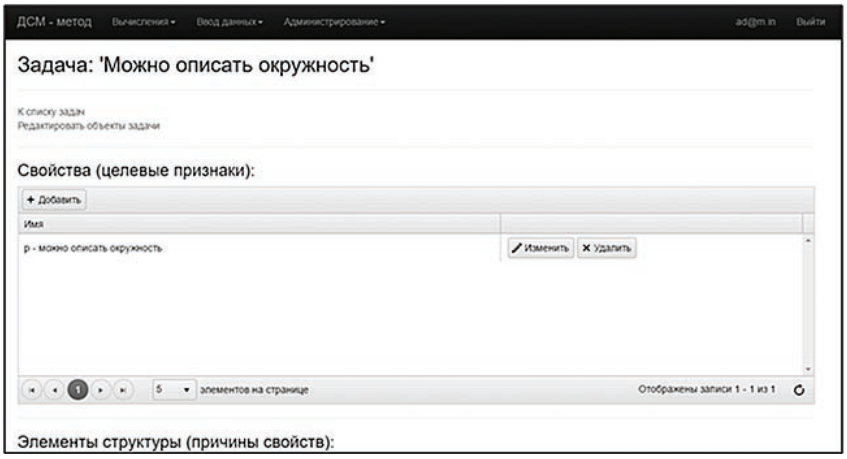


Рис. 8. Страница управления задачами



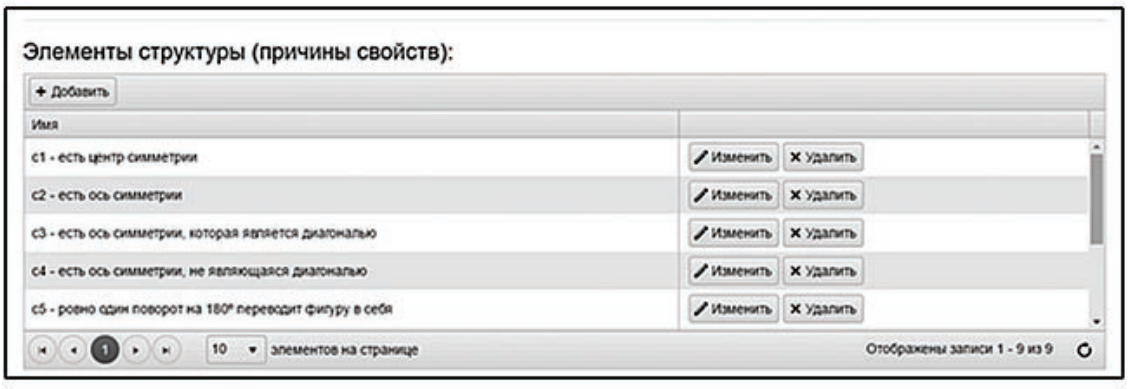


Рис. 10. Таблица редактирования структурных свойств

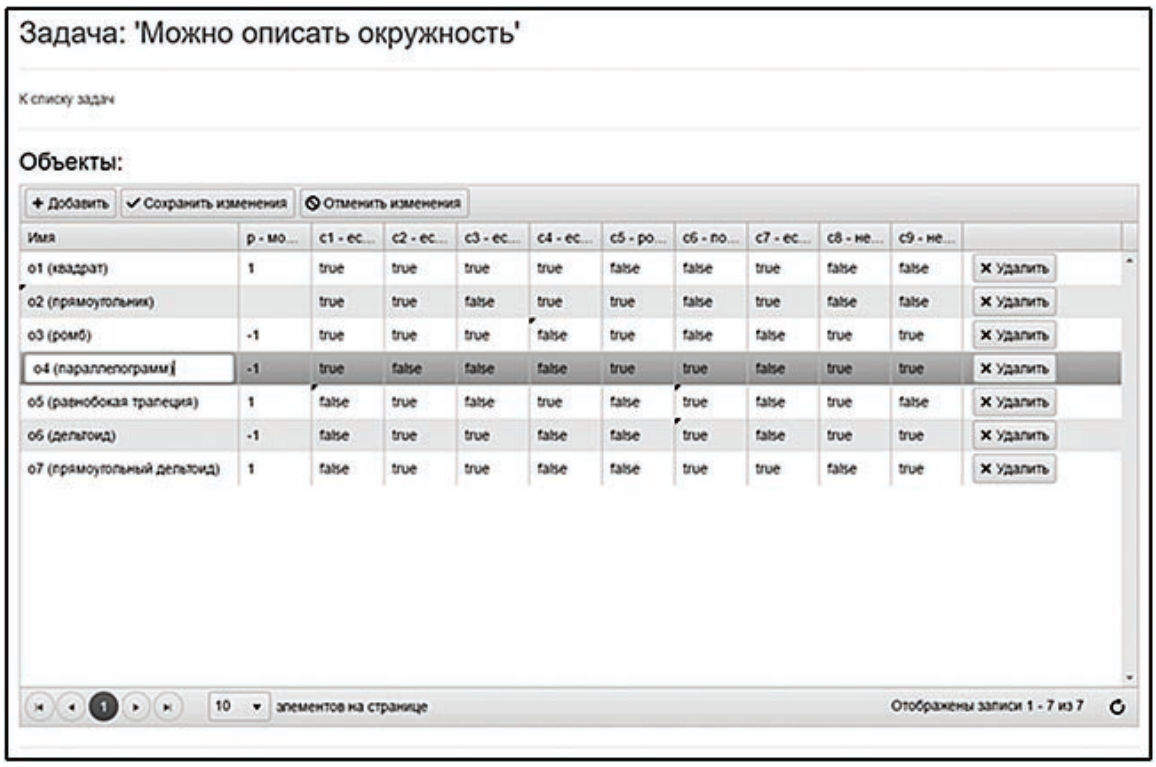


Рис. 11. Страница редактирования объектов задачи

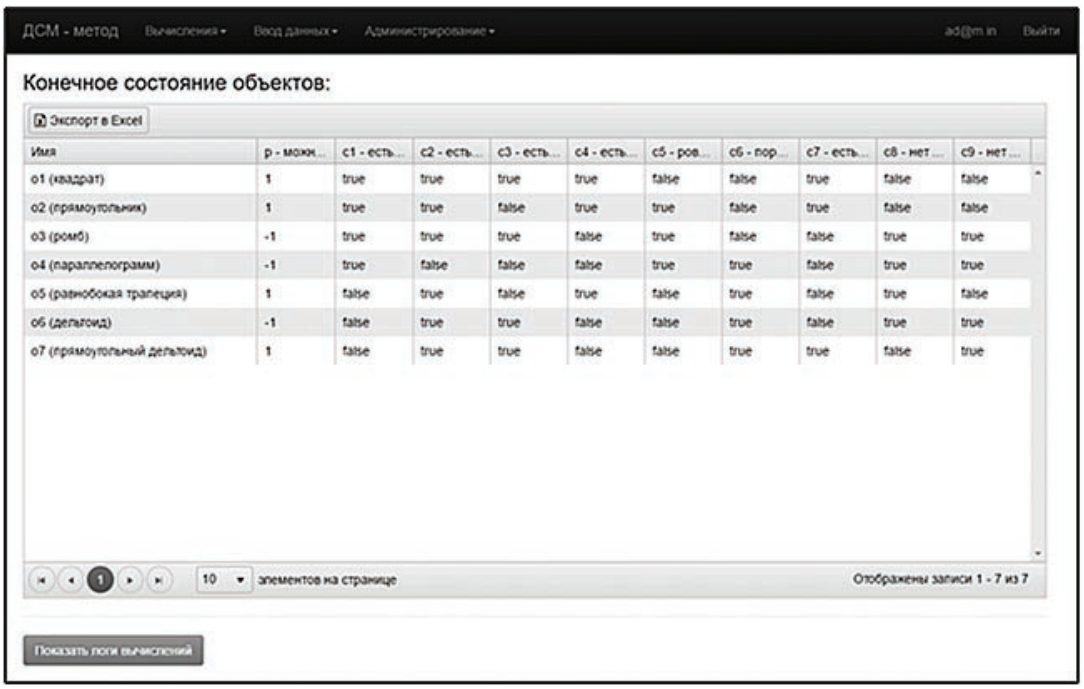


Рис. 12. Таблица с конечным состоянием объектов

рекомендуется во избежание замешательств у пользователей. Ниже на этой же странице размещена таблица для редактирования структурных элементов

задачи (рис. 10).

Далее обратим внимание на страницу редактирования объектов. Тут в таблице первый столбец — это название объекта, со второго до предпоследнего — по очереди сначала все свойства, потом все структурные признаки, последний столбец занимает кнопка для удаления элемента. В этой таблице поддерживается массовое редактирование элементов. Ячейки, которые были отредактированы недавно, помечаются красным треугольником, при нажатии на кнопку «Сохранить изменения» на редактирование отправляются сразу все элементы, которые были изменены. Значения структурных свойств могут быть true или false (во время редактирования появляется галочка). Значения свойств в соответствии с ДСМ-методом: 1, -1, 0 либо не иметь значения. Отсутствие значения подразумевает, что его необходимо найти. Также можно вернуть предыдущие значения ячеек, нажав на кнопку «Отменить изменения» (рис. 11).

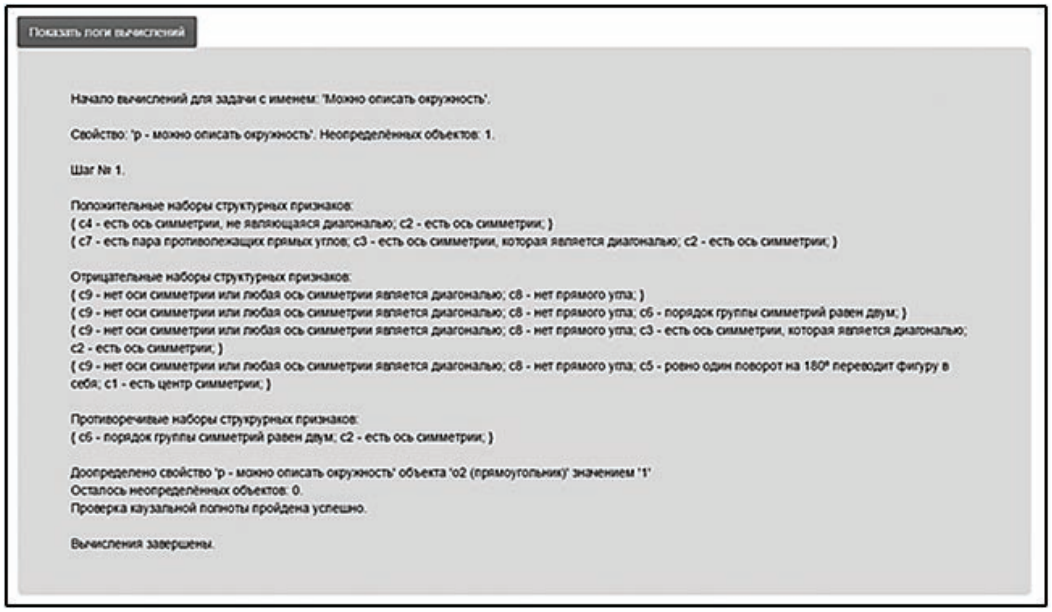


Рис. 13. Записи в лог

Теперь, когда данные добавлены в систему, можно переходить к вычислениям. Страница со списком задач для старта вычисления во многом похожа на страницу редактирования задач, поэтому приводиться не будет. Там при клике на название задачи осуществляется переход к результатам вычислений. На странице результатов вычислений есть таблица с конечным состоянием объектов, обратите внимание, что свойство у прямоугольника определено как значение один, что совпадает с рассматриваемым примером (рис. 12).

Тут же есть кнопка выгрузки результатов в Excel. Внизу, под таблицей, мы видим кнопку «Показать логи вычислений». Выгружаемые в лог записи были подробно рассмотрены во втором параграфе. Посмотрим, какие записи были сделаны во время работы над этой задачей (рис. 13).

Записи в лог. При клике по кнопке происходит разворачивание логов. Как и в рассмотренном примере, эта задача решается за один шаг ДСМ-метода.

Сгенерированные гипотезы совпадают, и причина свойства прямоугольника тоже: можно описать окружность, так как есть ось симметрии, не являющаяся диагональю. Есть запись о том, что проверка каузальной полноты пройдена. Таким образом, было продемонстрировано поведение системы со стороны пользователя, сгенерированы гипотезы на основе базы фактов и найдены значения свойств для всех объектов.

4. Инструкция для администратора по ДМС-программе

Зависимости

1) Microsoft .NET Framework 4.6.2. Скачать можно тут: https://www.microsoft.com/ru-ru/download/details.aspx?id=53344

2) IIS. Необходимо активировать IIS, создать на нём web-приложение. Инструкции есть в интернете, например тут: https://professorweb.ru/my/ASP_NET/sites/level3/3_1.php.

Развертывание и настройка

1) В корень web-приложения (например, C:\inetpub\wwwroot) поместить запускаемые файлы web-приложения. Найти их можно по ссылке:

https://drive.google.com/drive/folders/0B2T4JyuK3H-xYW5FaXVablNUejQ?usp=sharing.

2) В файле конфигурации веб-приложения Web.config (например, C:\inetpub\wwwroot\Web.config) указать строку подключения к базе данных DefaultConnection. Базу данных можно создать отдельно или подключить файл; пример строки, если подключаем файл:

<add name="DefaultConnection" connectionString="Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=

C:\inetpub\wwwroot\App_Data\JSMethod.mdf;Integrated Security=True;Connect Timeout=30"providerName="System.Data.SqlClient" />

3) Перезапустить web-приложение.

4) Теперь оно доступно по адресу: http://127.0.0.1 с web-сервера. Чтобы оно было доступно с других компьютеров сети, необходимо обращаться по ip сервера и порту web-приложения либо создать соответствующее сопоставление на DNS-сервере.

Где взять (скачать) приложение?

Приложение доступно либо по адресу:

`http://fkn.omsu.ru/Programs/JSM/workspace.zip`,

либо

`http://fkn.univer.omsk.su/workspace.zip`.

Как получить доступ на сайте?

Права администратора есть у учётной записи. Имя пользователя: ad@m.in Пароль: ARBGs5DjuTFv?5F7. Можно регистрировать сколько угодно новых пользователей и при помощи администраторской УЗ давать им доступ.

5. Другие компьютерные ДСМ-системы

1. ДСМ-система JSM-Socio Анны Волковой. Это интеллектуальная компьютерная система, предназначенная для решения задачи качественного анализа анкет социологического опроса с использованием стратегий ДСМ-метода: 1) извлечение причинно-следственных зависимостей, содержащихся в данных; 2) порождение гипотез с представлением описания субъекта, его мнения и ситуации; прогнозирование значения изучаемого эффекта; 3) оценка полученных гипотез посредством процедуры объяснения исходного множества фактов [32].

Система выдает результат, который представляет собой гипотезы, полученные различными стратегиями, и прогнозирование значения изучаемого эффекта у респондентов, для которых это значение изначально не определено.

2. ДСМ-система Ж.И. Бурковской. Это многоуровневая система с клиент-серверной архитектурой [32, с. 190].

В системе:

– программно реализованы различные инструменты формирования исходной базы фактов;

– разработан новый подход к проведению закрытых опросов общественного мнения (формирование мнений с помощью аргументационного оценивания);

– разработан модуль изучения рациональности мнений.

3. ДСМ-система Т.Л. Феофановой. Система создана на языке Prolog и позволяет ознакомить как эксперта-социолога, так и студента с работой самого метода [32, с. 191], [33].

Заключение

В статье представлено web-приложение, реализующее ДСМ-метод и позволяющее генерировать новые знания путём автоматического порождения гипотез из массивов уже имеющихся эмпирических знаний.

В какой мере данное приложение способно стать эффективным инструментом для исследователей? Хотелось бы надеяться, что найдутся социологи, которые в содружестве с компьютерщиками сумеют освоить предложенное web-приложение и применить его для анализа социологических данных и поиска зависимостей в опросах.

Важно, что в приложении редактировать данные могут только зарегистрированные пользователи (после выделения им специальных прав). За счёт разделения доступа при работе с web-приложением обеспечивается безопасность и защита от случайных ошибок пользователей и намеренной порчи данных.

Литература

1. Финн В.К. Индуктивные методы Д.С. Милля в системах искусственного интеллекта. Часть I // Искусственный интеллект и принятие решений. № 3, 2010. С. 3–21.
2. Финн В.К., Михеенкова М.А. О логических средствах концептуализации анализа мнений // Научно-техническая информация. Сер. 2. 2002. № 6. С. 4–22.
3. Гуц А.К. Математическая логика для социологов: учебное пособие. Омск : изд-во ОмГУ, 2017. 196 с.
4. Кученкова А.В. Логико-комбинаторные методы анализа социологических данных: эвристический потенциал и методическая специфика: автореф. ... канд. социол. наук. М. : РГГУ, 2012. 25 с.
5. Белозеров Д.М. Веб-приложение, реализующее ДСМ-метод: магистр. дисс. Омск : ОмГУ, 2017. 42 с.
6. ДСМ-метод // Википедия. URL: `http://ru.wikipedia.org/?oldid=83891713` (дата обращения: 27.05.2017).
7. Финн В.К. Индуктивные методы Д.С. Милля в системах искусственного интеллекта. Часть II // Искусственный интеллект и принятие решений. 2010. № 4. С. 14–40.
8. Финн В.К. Об определении эмпирических закономерностей посредством ДСМ-метода автоматического порождения гипотез // Искусственный интеллект и принятие решений. 2010. № 3. С. 41–48.
9. Новые возможности Visual Studio 2017 | Обновления продукта и новости / Официальный сайт Microsoft. URL: `https://www.visualstudio.com/ru/vs/whatsnew/` (дата обращения: 27.05.2017).
10. Agile, Git, CI with TFS | Team Foundation Server / Visual Studio IDE. URL: `https://www.visualstudio.com/ru/` (дата обращения: 27.05.2017).
11. Окончание поддержки Internet Explorer / Официальный сайт Microsoft. URL: `https://www.microsoft.com/ru-ru/windowsforbusiness/end-of-ie-support` (дата обращения: 27.05.2017).
12. ASP.NET MVC / The ASP.NET Site. URL: `https://www.asp.net/mvc` (дата обращения: 27.05.2017).
13. Razor Syntax Reference / Microsoft Docs. URL: `https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor` (дата обращения: 27.05.2017).
14. Bootstrap · The world's most popular mobile-first and responsive front-end framework. URL: `http://getbootstrap.com/` (дата обращения: 27.05.2017).
15. jQuery. URL: `https://jquery.com/` (дата обращения: 27.05.2017).
16. jQuery.ajax() / jQuery API Documentation. URL: `http://api.jquery.com/jquery.ajax/` (дата обращения: 27.05.2017).
17. Kendo UI / Telerik. URL: `http://www.telerik.com/kendo-ui` (дата обращения: 27.05.2017).
18. jQuery Grid control example / Kendo UI Grid Demos. URL: `http://demos.telerik.com/kendo-ui/grid/index` (дата обращения: 27.05.2017).
19. JSZip / GitHub. URL: `https://stuk.github.io/jszip/` (дата обращения: 27.05.2017).
20. Бандлы и минификация / METANIT.COM. URL: `https://metanit.com/sharp/mvc/15.1.php` (дата обращения: 27.05.2017).
21. Entity Framework Code-First / Entity Framework Tutorial. URL: `http://www.entityframeworktutorial.net/code-first/`

entity-framework-code-first.aspx (дата обращения: 27.05.2017).
22. ASP.NET Identity / The ASP.NET Site. URL: https://www.asp.net/identity (дата обращения: 27.05.2017).
23. Role based Authorization / Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles (дата обращения: 27.05.2017).
24. Генерация исходящих адресов URL / METANIT.COM. URL: https://metanit.com/sharp/mvc/6.4.php (дата обращения: 27.05.2017).
25. POST (HTTP) / Википедия. URL: http://ru.wikipedia.org/?oldid=83895724 (дата обращения: 27.05.2017).
26. Асинхронные методы, async и await | C♯ / METANIT.COM — Сайт о программировании. URL: https://metanit.com/sharp/tutorial/13.3.php (дата обращения: 27.05.2017).
27. Метод ObjectQuery(T).Include (String) (System.Data.Objects) / MSDN. URL: https://msdn.microsoft.com/ru-ru/library/bb738708(v=vs.110).aspx (дата обращения: 27.05.2017).
28. JSON. URL: http://www.json.org/json-ru.html (дата обращения: 27.05.2017).
29. Адаптивная вёрстка: что это и как использовать / Типичный программист. URL: https://tproger.ru/translations/responsive-web-design-tips/ (дата обращения: 27.05.2017).
30. Реализация базовой CRUD-функциональности с Entity Framework в приложении ASP.NET MVC / MSDN. URL: https://blogs.msdn.microsoft.com/vyunev/2011/10/05/crud-11/ (дата обращения: 27.05.2017).
31. Структура Guid (System) / MSDN. URL: https://msdn.microsoft.com/ru-ru/library/system.guid(v=vs.110).aspx (дата обращения: 27.05.2017).
32. Волкова А.Ю. Разработка алгоритмических и программных средств для реализации стратегий ДСМ-метода автоматического порождения гипотез: дис. ... канд. техн. наук. М. : РГГУ, 2014. 304 с.
33. Михеенкова М.А., Феофанова Т.Л. Обучающая ДСМ-система для анализа социологических данных // Вестник Российского государственного гуманитарного университета. 2009. Вып. 10. С. 152–169.

PROGRAM WEB-REALIZATION OF JSM-METHOD OF AUTOMATIC HYPOTHESIS GENERATION BASED ON INDUCTIVE LOGIC

D.M. Belozerov
Master, e-mail: evelend1@gmail.com
A.K. Guts
Dr.Sc. (Phys.-Math.), Professor, e-mail: guts@omsu.ru

Dostoevsky Omsk State University

Abstract. In the article, web-application that implements Finn's JSM method is presented. The application based on the Mill's inductive logic extracts new knowledge from the existing knowledge base.

Keywords: JSM-method, Mill inductive logic, web application.

References

1. Finn V.K. Induktivnye metody D.S. Millya v sistemakh iskusstvennogo intellekta. Chast' I. Iskusstvennyi intellekt i prinyatie reshenii, no.3, 2010, pp. 3-21. (in Russian)
2. Finn V.K. and Mikheenkova M.A. O logicheskikh sredstvakh kontseptualizatsii analiza mnenii. Nauchno-tekhnicheskaya informatsiya, Cer. 2, 2002, no. 6, pp. 4–22. (in Russian)
3. Guts A.K. Matematicheskaya logika dlya sotsiologov: uchebnoe posobie. Omsk, OmGU Publ., 2017, 196 p. (in Russian)
4. Kuchenkova A.V. Logiko-kombinatornye metody analiza sotsiologicheskikh dannykh: evristicheskii potentsial i metodicheskaya spetsifika: avtoref. ... kand. sotsiol. nauk. Moscow, RGGU, 2012, 25 p. (in Russian)
5. Belozerov D.M. Veb-prilozhenie, realizuyushchee DSM-metod: magistr. diss. Omsk, OmGU, 2017, 42 p. (in Russian)
6. DSM-metod, Vikipediya. URL: http://ru.wikipedia.org/?oldid=83891713. (in Russian)
7. Finn V.K. Induktivnye metody D.S. Millya v sistemakh iskusstvennogo intellekta. Chast' II. Iskusstvennyi intellekt i prinyatie reshenii, 2010, no.4, pp. 14–40. (in Russian)
8. Finn V.K. Ob opredelenii empiricheskikh zakonomernostei posredstvom DSM-metoda avtomaticheskogo porozhdeniya gipotez. Iskusstvennyi intellekt i prinyatie reshenii, 2010, no.3, pp. 41–48. (in Russian)
9. Novye vozmozhnosti Visual Studio 2017 — Obnovleniya produkta i novosti. Ofitsial'nyi sait Microsoft. URL: https://www.visualstudio.com/ru/vs/whatsnew/. (in Russian)
10. Agile, Git, CI with TFS — Team Foundation Server. Visual Studio IDE. URL: https://www.visualstudio.com/ru/. (in Russian)
11. Okonchanie podderzhki Internet Explorer. Ofitsial'nyi sait Microsoft. URL: https://www.microsoft.com/ru-ru/windowsforbusiness/end-of-ie-support. (in Russian)
12. ASP.NET MVC. The ASP.NET Site. URL: https://www.asp.net/mvc.
13. Razor Syntax Reference. Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor.
14. Bootstrap · The world's most popular mobile-first and responsive front-end framework. URL: http://getbootstrap.com/.
15. jQuery. URL: https://jquery.com/.
16. jQuery.ajax(). jQuery API Documentation. URL: http://api.jquery.com/jquery.ajax/.
17. Kendo UI. Telerik. URL: http://www.telerik.com/kendo-ui.
18. jQuery Grid control example. Kendo UI Grid Demos. URL: http://demos.telerik.com/kendo-ui/grid/index.
19. JSZip. GitHub. URL: https://stuk.github.io/jszip/. (in Russian)

20. Bandly i minifikatsiya. METANIT.COM. URL: https://metanit.com/sharp/mvc/15.1.php. (in Russian)
21. Entity Framework Code-First. Entity Framework Tutorial. URL: http://www.entityframeworktutorial.net/code-first/entity-framework-code-first.aspx.
22. ASP.NET Identity. The ASP.NET Site. URL: https://www.asp.net/identity.
23. Role based Authorization. Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles.
24. Generatsiya iskhodyashchikh adresov URL. METANIT.COM. URL: https://metanit.com/sharp/mvc/6.4.php. (in Russian)
25. POST (HTTP). Vikipediya. URL: http://ru.wikipedia.org/?oldid=83895724. (in Russian)
26. Asinkhronnye metody, async i await — C♯. METANIT.COM — Sait o programmirovanii. URL: https://metanit.com/sharp/tutorial/13.3.php. (in Russian)
27. Metod ObjectQuery(T).Include (String) (System.Data.Objects). MSDN. URL: https://msdn.microsoft.com/ru-ru/library/bb738708(v=vs.110).aspx.
28. JSON. URL: http://www.json.org/json-ru.html. (in Russian)
29. Adaptivnaya verstka: chto eto i kak ispol'zovat'. Tipichnyi programmist. URL: https://tproger.ru/translations/responsive-web-design-tips/. (in Russian)
30. Realizatsiya bazovoi CRUD-funktsional'nosti s Entity Framework v prilozhenii ASP.NET MVC. MSDN. URL: https://blogs.msdn.microsoft.com/vyunev/2011/10/05/crud-11/. (in Russian)
31. Struktura Guid (System). MSDN. URL: https://msdn.microsoft.com/ru-ru/library/system.guid(v=vs.110).aspx. (in Russian)
32. Volkova A.Yu. Razrabotka algoritmicheskikh i programmnykh sredstv dlya realizatsii strategii DSM-metoda avtomaticheskogo porozhdeniya gipotez: dis. ... kand. tekhn. nauk. Moscow, RGGU, 2014, 304 p. (in Russian)
33. Mikheenkova M.A. and Feofanova T.L. Obuchayushchaya DSM-sistema dlya analiza sotsiologicheskikh dannykh. Vestnik Rossiiskogo gosudarstvennogo gumanitarnogo universiteta, 2009, Vyp. 10, pp. 152–169. (in Russian)

Дата поступления в редакцию: 03.10.2020