Методы визуализации больших деревьев, возникающих при решении задач оптимизации методом ветвей и границ

М.А. Посыпкин, А.Л. Усов

***Аннотация*—в данной работе рассматривается способ сохранения, анализа и визуализации деревьев размером 100 млн. узлов и более возникающих при решении задач оптимизации методом ветвей и границ. В решении предлагается способ сбора первичных данных от решающей программы, анализ и обработка этих данных, а также их последующая визуализация. В статье описывается архитектура и основные программные модули предлагаемого решения.**

***Ключевые слова*—визуализация дерева, задача оптимизации, метод ветвей и границ, обобщенное дерево.**

I. Введение

В настоящий момент существует несколько программ для визуализации алгоритма метода ветвей и границ. В частности был исследован инструмент для визуализации деревьев Branch and Bound Analysis Kit [1] и некоторые базовые идеи были использованы в настоящей работе.

Ключевым отличием разработанного программного комплекса является возможность визуализации обобщенных деревьев с большим количеством узлов. Если принять, что для хранения информации по каждому узлу дерева необходимо 150 байт, то, например, для хранения дерева из 100 млн. узлов уже потребуется порядка 15 ГБ. Становится понятным, что манипуляции с подобными деревьями в оперативной памяти пользовательского компьютера будут затруднительны. Так же возникает проблема отображения большого дерева на экране компьютера. При стандартном разрешении 1280x1024 пикселей на экране можно отобразить не более чем 150 узлов дерева при минимальном расстоянии между узлами 96 пикселей.

Для решения этих проблем была разработана архитектура, особенности которой описаны в данной работе. Результаты экспериментов при работе с деревьями разных размеров представлены в конце статьи.

Статья получена 15 июня 2016.

М.А. Посыпкин работает в ВЦ ФИЦ ИУ РАН (e-mail: mposypkin@gmail.com).

А.Л. Усов является студентом ВВО ВМК МГУ (alusov@mail.ru).

Работа поддержана грантом министерства образования и науки республики Казахстан, проект 0115PK00554.

II. Ключевые понятия

Решающая программа (B&B solver) – программа, вычисляющая оптимизационную задачу методом ветвей и границ.

Обобщенное дерево (Generalized tree) – дерево, не имеющее ограничений на количество потомков которое может иметь каждый узел. ([5])

Первый потомок (First child) – первый потомок узла обобщенного дерева. Потомки узла дерева связаны между собой при помощи двунаправленного списка. При этом первый потомок является головным узлом списка. Организация хранения дерева проиллюстрирована на Рис.1-2.

Родительский узел (Parent node) – ближайший предок узла дерева.

Левый узел (Left sibling) – левый узел относительно данного узла в связанном списке потомков.

Правый узел (Right sibling) – правый узел относительно данного узла в связанном списке потомков.

Уровень узла – определяется следующим образом: корень дерева имеет уровень 0, а уровень любого другого узла на 1 больше уровня родительского узла.

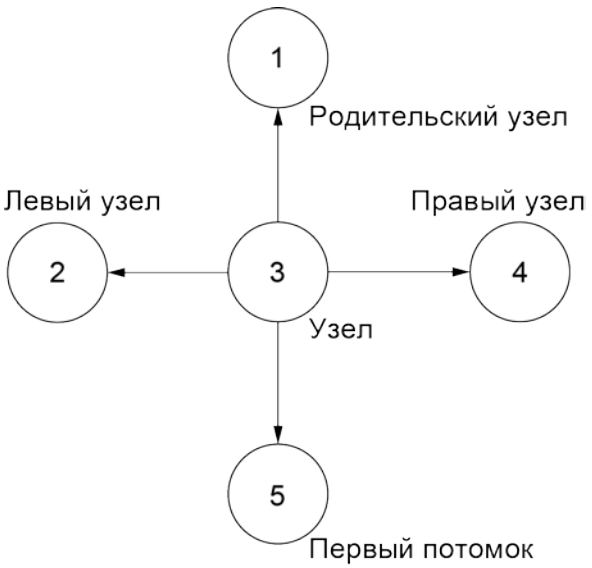

Рис. 1. Ссылки узла в обобщенном дереве

III. Сбор первичных данных от решающей программы

Для сбора первичных данных от решающей программы был разработан формат хранения (Таб. 1). Каждая запись о вновь созданном узле сохраняется в текстовом формате с предопределенной длиной полей. Запись имеет четко заданную длину, что позволяет быстро найти любой узел по его идентификатору без перебора всех записей. Текстовый формат хранения был выбран из-за его простоты и возможности непосредственного просмотра данных. В разработанном формате возможно сохранить данные от независимо работающей решающей программы без использования API (Рис. 3) и передать их исследователю для последующего анализа.

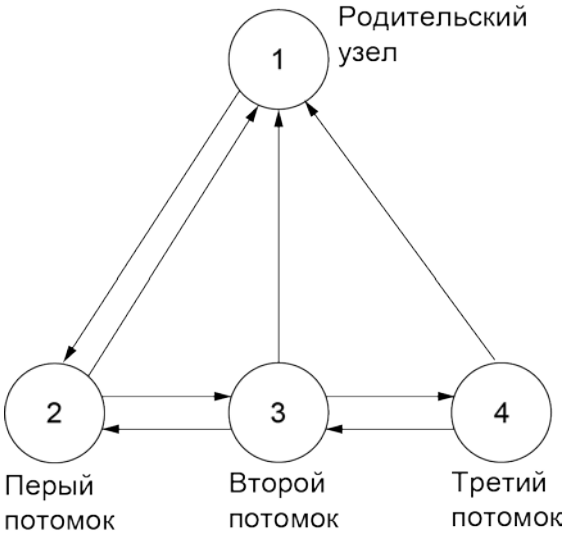

Рис. 2. Двусвязанный список потомков узла 1. Первый потомок узел 2 является началом связанного списка.

В тоже время для удобства сбора первичных данных был разработан API в виде функции обратного вызова. Для слабой связанности программных модулей решающей программе передается функция обратного вызова int CreateNode(int parentId, ...) при помощи, которой генерируется событие о создании нового узла. Функция возвращает решающей программе автоматически сгенерированный идентификатор узла дерева. Реализация функции выполнена при помощи модуля работы с хранилищем (Рис. 3). Задача этого модуля - накапливать события в кольцевом буфере и сохранять данные в текстовый файл по мере его заполнения (Таб. 1).

Узлы дерева нумеруются подряд, начиная с единицы. Номера потомков родительского узла так же нумеруются подряд, начиная с единицы. Решающая программа использует получаемый идентификатор узла для последующей передачи в функцию int CreateNode(int parentId, ...) в качестве идентификатора родительского узла. При этом корневой узел дерева имеет родительский узел равный нулю.

Таб. 1 Формат хранения первичных данных

Время события	Идентификатор узла	Идентификатор родительского узла	Номер потомка	Состояние узла	Любая информация связанная с узлом
18:27:05:875	1	0	1	3	Данные узла 1
18:27:05:876	2	1	1	3	Данные узла 2

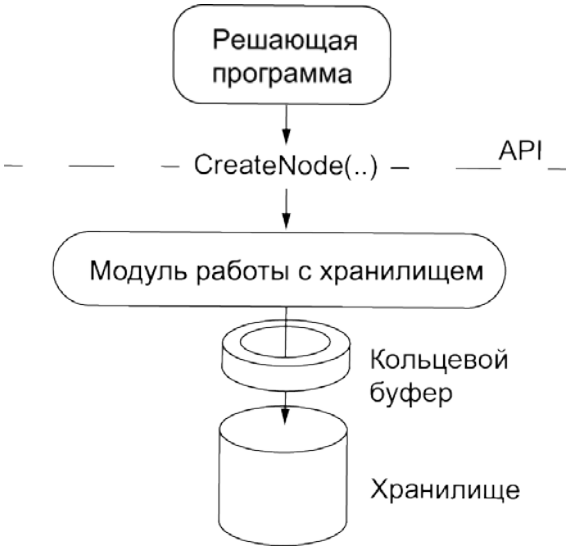

Рис. 3. Сохранение данных генерируемых решающей программой

IV. Анализ и обработка первичных данных

После получения первичных данных от решающей программы данные проходят обработку при помощи трех программных модулей (Рис. 4).

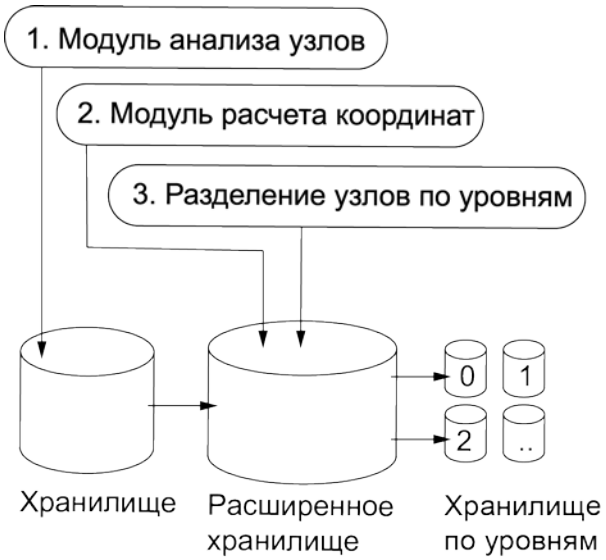

Рис. 4. Основные программные модули

A. Модуль анализа узлов дерева

Задача этого модуля скопировать каждую запись в расширенное хранилище и добавить для каждого узла идентификаторы левого узла, правого узла и первого потомка (Таб. 2). Расширенное хранилище организовано для возможности расширения данных по каждому узлу дерева, при этом первичное хранилище используется в режиме только для чтения. Алгоритм работает следующим образом: в цикле по каждой записи читается идентификатор родительского узла и номер потомка. Если номер потомка равен 1, то в родительский узел записывается идентификатор первого потомка. Если номер потомка больше 1, то в родительском узле читается идентификатор первого потомка, который является головным элементом двусвязного списка (Рис. 2). Далее алгоритм доходит до конца этого списка т.е когда идентификатор правого узла равен 0 и добавляет потомок в конец. При этом у последнего потомка сохраняется идентификатор левого узла, а у предпоследнего идентификатор правого узла соответственно.

Следует отметить, что вся последующая работа с деревом осуществляется только через файл. Это связано с тем, что предполагаемые размеры дерева могут быть очень большими и ресурса оперативной памяти компьютера пользователя не хватит для работы с подобным деревом. При работе с файлом используется перемещение указателя по файлу, а выделение оперативной памяти требуется только для текущих обрабатываемых узлов. В данной работе принято соглашение, что нулевой идентификатор узла соответствует пустому или, иначе говоря, идентификатор отсутствует.

Таб. 2 Формат расширенного хранилища

Первичные поля узла (Таб. 1)	Левый узел	Правый узел	Первый потомок	Поля модуля расчета координат
...	0	0	2	...
...	3	5	10	...
...	...	...	...	...

B. Модуль Расчета координат

После отработки модуля анализа узлов в работу включается модуль расчета координат (Рис. 4). Его цель рассчитать координаты X и Y для всех узлов дерева. При этом за начало координат принимается корень дерева. Координаты узлов дерева хранятся в условных единицах. Минимально возможное расстояние между двумя узлами принято равной одной условной единице. Расчет координаты Y осуществляется путем обхода дерева в прямом порядке (pre-order walk), используя данные о первом потомке и правом узле. Координата Y каждого узла дерева соответствует уровню узла. Расчет координаты X основан на алгоритме описанном в статье [4] и на исследовании работ [2] и [3]. Отображение узлов дерева осуществляется на эстетических правилах описанных в статье [3]:

- Узлы одного уровня дерева находиться на прямой линии. Прямые линии определяющие уровни должны быть параллельны.
- Родительский узел должен находиться посередине над его потомками.
- Дерево и его зеркальное изображение должны отрисовываться как отражение друг друга. Поддерево должно быть отображено одинаково независимо от того где оно расположено в дереве.
- Небольшие внутренние поддеревья должны располагаться равномерно среди больших поддеревьев, если большие деревья являются смежными между собой на одном или более уровне.
- Небольшие поддеревья крайние слева или справа должны быть смежными с большими поддеревьями.

Алгоритм реализованный в модуле расчета координат полностью соответствует этим правилам. Следует отметить, что сложность этого алгоритма линейная O(n). В частности, это достигается при помощи использования техники специальных указателей в узлах, описанных в статье [2] как “потоки”. Они необходимы для обхода контура поддеревьев при их взаимном позиционировании.

Поля необходимые для работы модуля расчета координат описаны в Таб. 3. Вспомогательные поля такие как предварительная координата X, изменение координаты, поток и предок необходимы для реализации алгоритма расчета координат X описанного в статье [4].

Таб. 3 Данные добавляемые модулем расчета координат

Первичные поля узла	Поля модуля анализа узлов	X	Y	Предварительная координата X	Изменение координаты	Поток	Предок
...	...	0	0	2.88	0	0	1
...	...	-1.62	1	1.25	0	0	2
...	...	...	...	...	...	...	...

C. Модуль разделения узлов по уровням

Модуль разделения узлов по уровням необходим для организации быстрого поиска узлов на любом уровне дерева. Алгоритм осуществляет обход дерева в левостороннем порядке и накапливает данные по узлам в буфере. Буфер реализован виде хеш-таблицы, где в качестве ключа используется уровень узла т.е координата Y, а в качестве значения список структур из двух полей – идентификатор узла и координата X. По мере накопления буфера эти данные сохраняются в текстовом файле в формате описанном в Таб. 4. Данные сохраняются в отдельный файл для каждого уровня дерева. При этом в имени файла используется уникальный номер уровня дерева. Координаты X узлов дерева хранятся в файле в виде возрастающего списка. Это позволяет найти любой диапазон узлов от заданной левой границы до правой используя бинарный поиск.

Таб. 4 Формат хранения координат X по уровням дерева

Идентификатор узла	X
5	-16.31
29	-11.56
34	-10.56
...	...

Кроме этого, данный модуль сохраняет распределение количества узлов по уровням дерева (Таб. 5). Эти данные позволяют построить график распределения узлов и увидеть признаки несбалансированности дерева. В статье [1] дан пример, когда одна из ветвей дерева опускается глубоко вниз практически без ветвления. В результате на графике будет видно, что на больших глубинах дерева расположено относительно малое количество узлов. Эта информация помогает в исследовании эффективности работы решающей программы.

Таб. 5 Формат хранения количества узлов по уровням дерева

Уровень дерева	Количество узлов	Количество листьев
0	1	0
1	1	0
2	2	1
...	...	...

V. Визуализация дерева

Визуализация дерева реализована в рамках архитектуры клиент-сервер (Рис. 5). В данной работе реализовано два основных сценария работы пользователя:

- Визуализация дерева по идентификатору узла. Пользователь вводит в поле ввода номер узла и выполняет запрос.
- Визуализация дерева по координате экрана. Пользователь при помощи мыши перемещает область просмотра дерева (Рис. 6).

В обоих сценариях серверу передаются текущие размеры окна, где изображается дерево. Следует отметить что, координаты точек экрана и его размеры вычисляются в пикселях. Для передачи этих данных серверу они преобразуются в условные единицы при помощи операции масштабирования и переноса. В работе принято соглашение: одна условная единица равна 96 пикселям, что соответствует минимальному расстоянию между двумя узлами дерева.

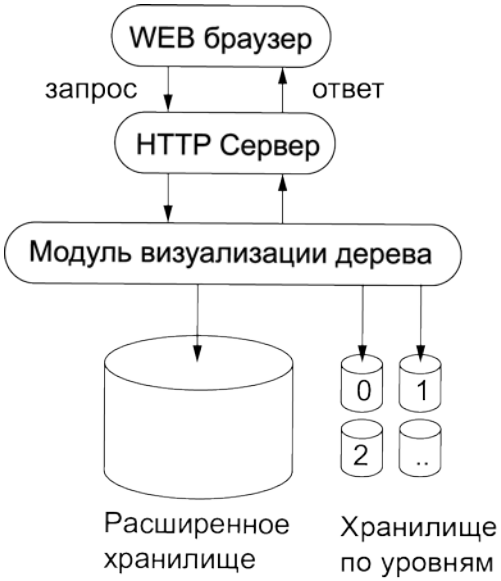

Рис. 5. Клиент серверная архитектура визуализации дерева

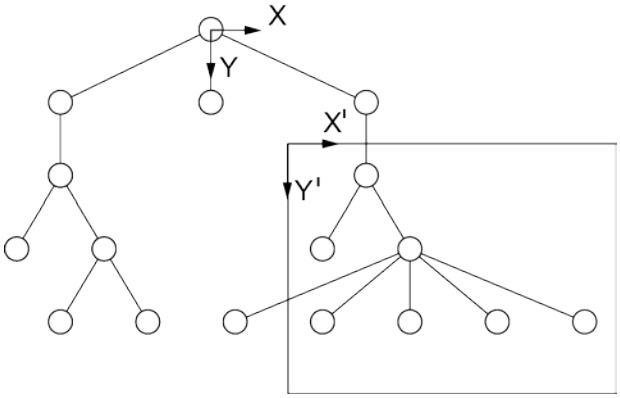

Рис. 6. Визуализация дерева на экране

HTTP сервер передает модулю визуализации дерева идентификатор узла и размеры окна. Модуль читает координаты X и Y узла из расширенного хранилища. Учитывая, что узел изображается в центре окна, вычисляется левая, правая, верхняя и нижняя границы области изображения. Верхняя и нижняя границы окна определяют уровни отображаемого дерева. Начиная с верхнего до нижнего уровня модуль ищет все узлы находящиеся между левой и правой границей окна. При помощи бинарного поиска модуль находит первый левый узел в хранилищах по уровням (Таб. 4). Далее в цикле считываются все идентификаторы узлов до правой границы. Из расширенного хранилища считываются все узлы соответсвующие найденным идентификаторам. Ребра между узлами вычисляются на

основе данных о потомках и о родительском узле. Окончательно все найденные узлы и ребра передаются клиентской стороне в JSON формате. Клиент делает преобразование условных координат в пикселы и визуализирует дерево. Для отображения дерева используется векторная графика в SVG формате.

Запрос пользователя по координате экрана работает следующим образом: пользователь щелчком левой кнопки мыши указывает начало вектора перемещения дерева, переместив мышь, он отпускает кнопку, тем самым указывая конец вектора. По этим данным расчитываются координаты X и Y центра отображаемой области.

На Рис. 7 показан пример визуализации дерева в Web-браузере. В верхней части экрана пользователь вводит идентификатор узла, а в нижней части отображается поддерево соответствующее этому идентификатору. При наведении указателя мыши на любой узел дерева, отображается всплывающая подсказка со всей информацией связанной с данным узлом.

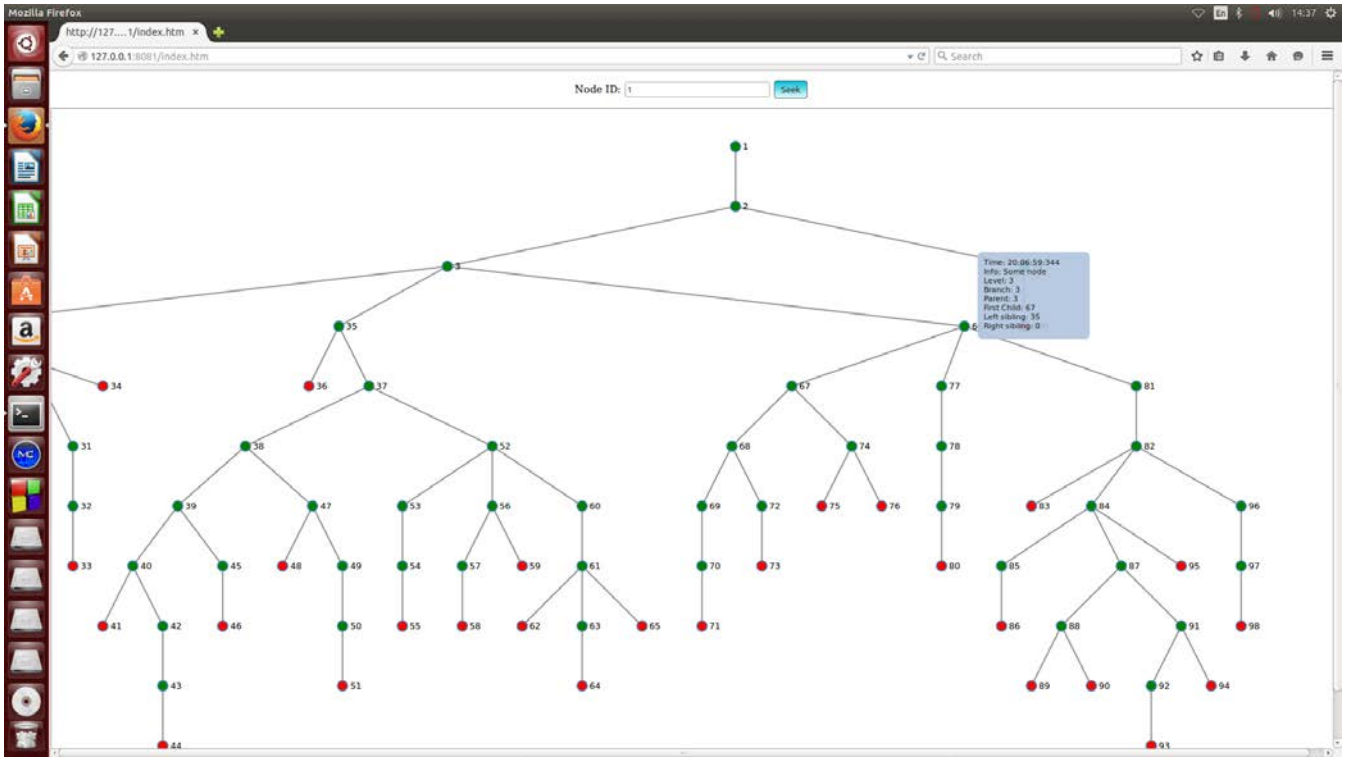

Рис. 7 Визуализация дерева в Web-браузере

Таб. 6. Результаты обработки деревьев

Количество узлов дерева	Время работы программных модулей			Общее время работы модулей	Среднее время поиска узлов для окна 1280x1024 пикселов
	Анализ узлов	Расчет координат	Разделение узлов по уровням		
1 млн.	10.28 с	43.57 с	14.34 с	68.19 с	0.002 с
10 млн.	107.18 с	462.45 с	149.26 с	718.89 с	0.002 с
100 млн.	1092.23	4845.34 с	1532.34 с	7469.91 с	0.003 с

VI. Результаты экспериментов при работе с деревьями разной размерности

В Таб. 6 представлены результаты измерения времени работы различных этапов обработки и визуализации деревьев на ноутбуке с процессором Intel Core i7-6700HQ CPU 2.6 GHz RAM 8Гб. Была протестирована работа программы для отображения дерева, содержащего 100 млн. узлов. При этом на некоторых уровнях дерева количество узлов достигало 10 млн. В среднем время обработки дерева увеличивается пропорционально количеству узлов дерева. Важно отметить, что при увеличении количества узлов на уровне в n раз, количество запросов в хранилище по уровням увеличивается на lg n. В нашем примере при увеличении количества узлов в 10 раз максимальное число запросов на уровне увеличивается с 23-24 раз до 26-27 раз, что мало влияет на время поиска узлов дерева. Кроме того, данная архитектура позволяет параллельное выполение запросов по каждому уровню дерева.

VII. Заключение

В работе был рассмотрен способ визуализации больших ветвлений деревьев возникающих при решении задач оптимизации методом ветвей и границ. Была рассмотрена программная реализация модулей обработки и визуализации дерева, а также форматы хранения данных. Проведенные эксперименты показали

эффективность предложенного подхода для визуализации деревьев до 100 млн. узлов.

Вместе с тем, при визуализации больших деревьев возможна ситуация когда ребро изображается практически горизонтально на экране т.к. разница между X координатами узлов на концах ребра значительно превышают разницу между Y координатами. Кроме этого, исследователю дерева необходима возможность отображения всего дерева на экране независимо от его размеров. Эти вопросы планируется решить при помощи масштабирования дерева.

Библиография

[1] Osman Y. Ozaltın, Brady Hunsaker. Visualizing Branch-and-Bound Algorithms. Industrial Engineering Department, University of Pittsburgh, Pittsburgh, PA 15261, USA. Ted K. Ralphs Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015-1582, USA. 2004.

[2] Edward M. Reingold and John S. Tilford. Tidier Drawings of Trees. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-7, NO. 2, March 1981.

[3] John Q. Walker II A Node-Positioning Algorithm for General Trees. The University of North Carolina at Chapel Hill Department of Computer Science CB#3175, Sitterson Hall Chapel Hill, NC 27599-3175. September, 1989.

[4] Cristoph Buchheim, Michael Junger, and Sebastian Leipert. Improving Walker's Algorithm to Run in Linear Time.

[5] Дональд Кнут. Искусство программирования, том 1. Основные алгоритмы = The Art of Computer Programming, vol.1. Fundamental Algorithms. — 3-е изд. — М.: «Вильямс», 2006.

Visualizing of large trees, resulting in solving optimization problems by branch and bound algorithm.

M.A. Posypkin, A.L. Usov

***Abstract*— In this paper, we consider a method of storage, analysis and visualizing of large trees with the amount of 100 million nodes and more resulting in solving optimization problems by branch and bound algorithm. The solution provides a method for the collection of primary data from solver, analysis and processing of this data, as well as their subsequent visualization. The paper describes the architecture and basic software modules of the proposed solution.**

Keywords— branch and bound algorithm, generalized tree, optimization problems, visualizing tree.