ЛАБОРАТОРНАЯ РАБОТА №2

«ПОИСК РЕШЕНИЙ CSP ЗАДАЧ»

1. Цель работы

Изучение особенностей задач удовлетворения ограничений (CSP - Constraint Satisfaction Problem) и исследование основных методов поиска их решений средствами языка Пролог.

2. Постановка задачи

2.1. Изучить методы представления и решения CSP задач по лекционному материалу и книгам.

2.2. Ознакомиться по лекционному материалу и книгам с объявлением операторов, с расширенным перечнем предикатов обработки списков, метаусловиями и отрицанием в языке Пролог. Изучить примеры применения этих средств Пролога для решения CSP задач, которые приведены в п. 2.2 настоящей лабораторной работы.

2.3. Ознакомиться с вариантом задания. Сформулировать задачу в терминах задач CSP (см. п. 2.2.1):

a) определить перечень переменных задачи;

б) специфицировать области определения каждой переменной;

в) определить отношения ограничения между переменными.

2.4. Ознакомиться с примером кода, приведенного в приложении Б, и, по аналогии, определить на языке Пролог необходимые предикаты для решения поставленной задачи.

2.5. Создать в среде программирования Пролог-проект и выполнить его отладку.

Исследовать свойства разработанной программы. Оценить количество просматриваемых сочетаний переменных, определить с помощью предиката time статистику выполнения программы. Сформулировать варианты улучшения программы. Предложить альтернативные методы решения задачи.

2.7. Зафиксировать результаты работы программы в виде экранных копий.

3. Вариант задания

Вариант 19. Дама сдавала в багаж: диван, чемодан, саквояж, картину, корзину, картонку и маленькую собачонку. Диван весил столько же, сколько чемодан и саквояж вместе, и столько же, сколько картина и картонка вместе. Картина, корзина и картонка весили поровну, причем каждая из них — больше, чем собачонка. Когда выгружали багаж, дама заявила, что собака не той породы. При проверке оказалось, что собака перевешивает диван, если к ней на весы добавить саквояж или чемодан. Требуется написать программу, которая доказала бы справедливость претензии дамы.

4. Ход работы

Для вышеприведенной задачи переменными являются диван, чемодан, саквояж, картина, корзина, картонка и маленькая собачонка. К этим переменным стоит добавить так же собаку другой породы.

Область определения каждой переменной пускай будет вес от 1 до 100 килограмм.

Множество ограничений следующее:

1) Диван = Чемодан + Саквояж;

2) Диван = Картина + Картонка; => Диван = 2\*Картина;

3) Картина = Корзина = Картонка;

4) Картина > Собачонка;

5) Собака другой породы + Саквояж > Диван;

6) Собака другой породы + Чемодан > Диван.

Листинг 1 – Код программы

solve :-

  Obj = [Sofa, Suitcase, Bag, Picture, Basket, Vase, SmallDog, BigDog],

  % Ограничения

  Sofa #= Suitcase + Bag,

  Sofa #= Picture + Vase,

  Picture #= Basket,

  Picture #= Vase,

  Picture #> SmallDog,

  BigDog + Bag #> Sofa,

  BigDog + Suitcase #> Sofa,

  % Поиск решения

  my\_label(Obj),

  % Вывод решения

  format('Sofa: ~d ~n', [Sofa]),

  format('Suitcase: ~d ~n', [Suitcase]),

  format('Bag: ~d ~n', [Bag]),

  format('Picture: ~d ~n', [Picture]),

  format('Basket: ~d ~n', [Basket]),

  format('Vase: ~d ~n', [Vase]),

  format('Small Dog: ~d ~n', [SmallDog]),

  format('Big Dog: ~d ~n', [BigDog]).

my\_label([]).

my\_label([Var | Vars]) :-

  between(1, 10, Var),

  my\_label(Vars).

%===============вспомогательный предикат========================================

% цикл повторения выполнения Цели заданное число раз (N)

rep(Aim,1):-Aim.

rep(Aim,N):-

not(not(Aim)), %стирание предыдущих подстановок

M is N-1,rep(Aim,M).

Разработанный код был протестирован. Результаты тестов изображены на рисунках 1-2.

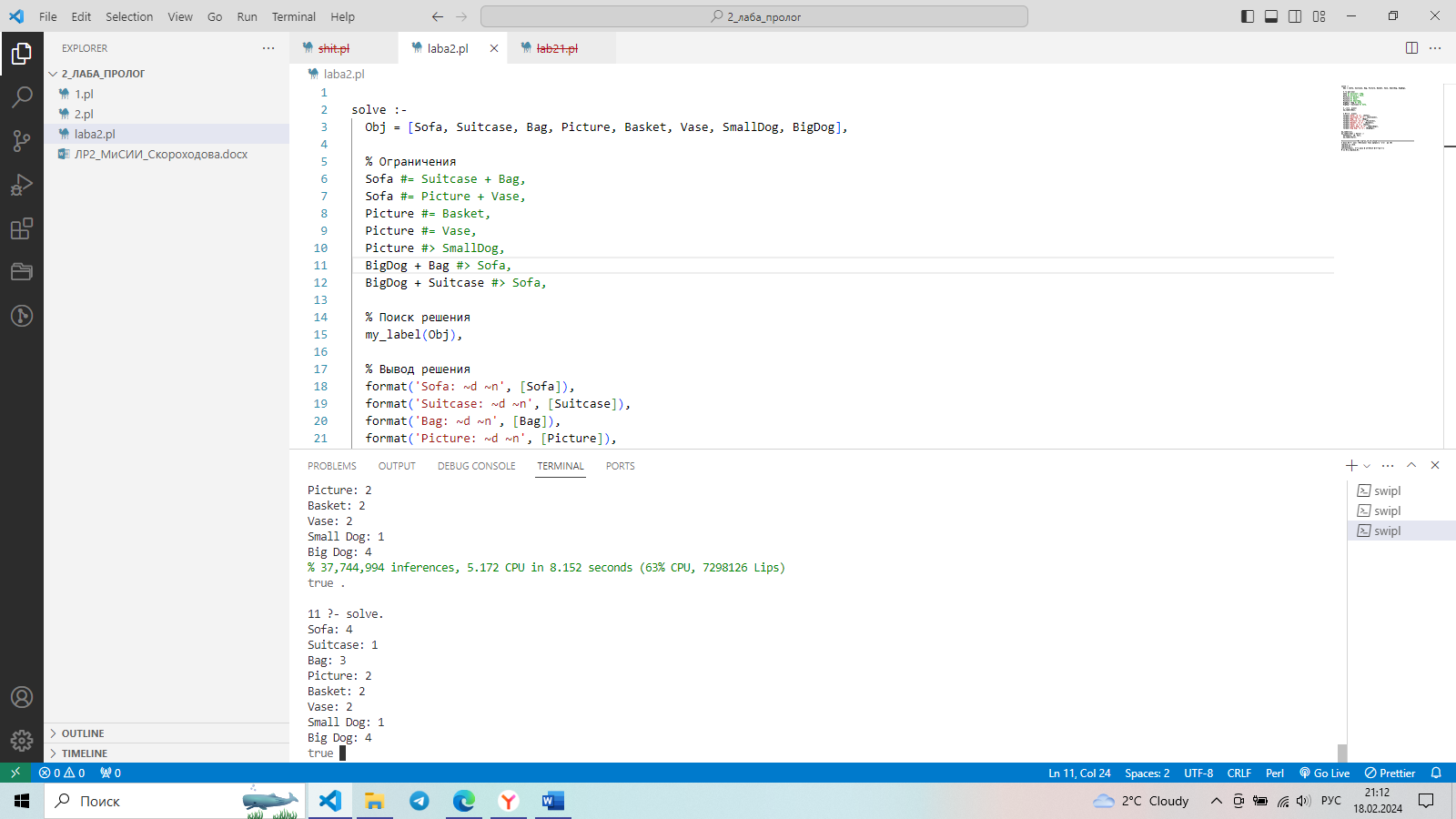


Рисунок 1 – Результаты решения

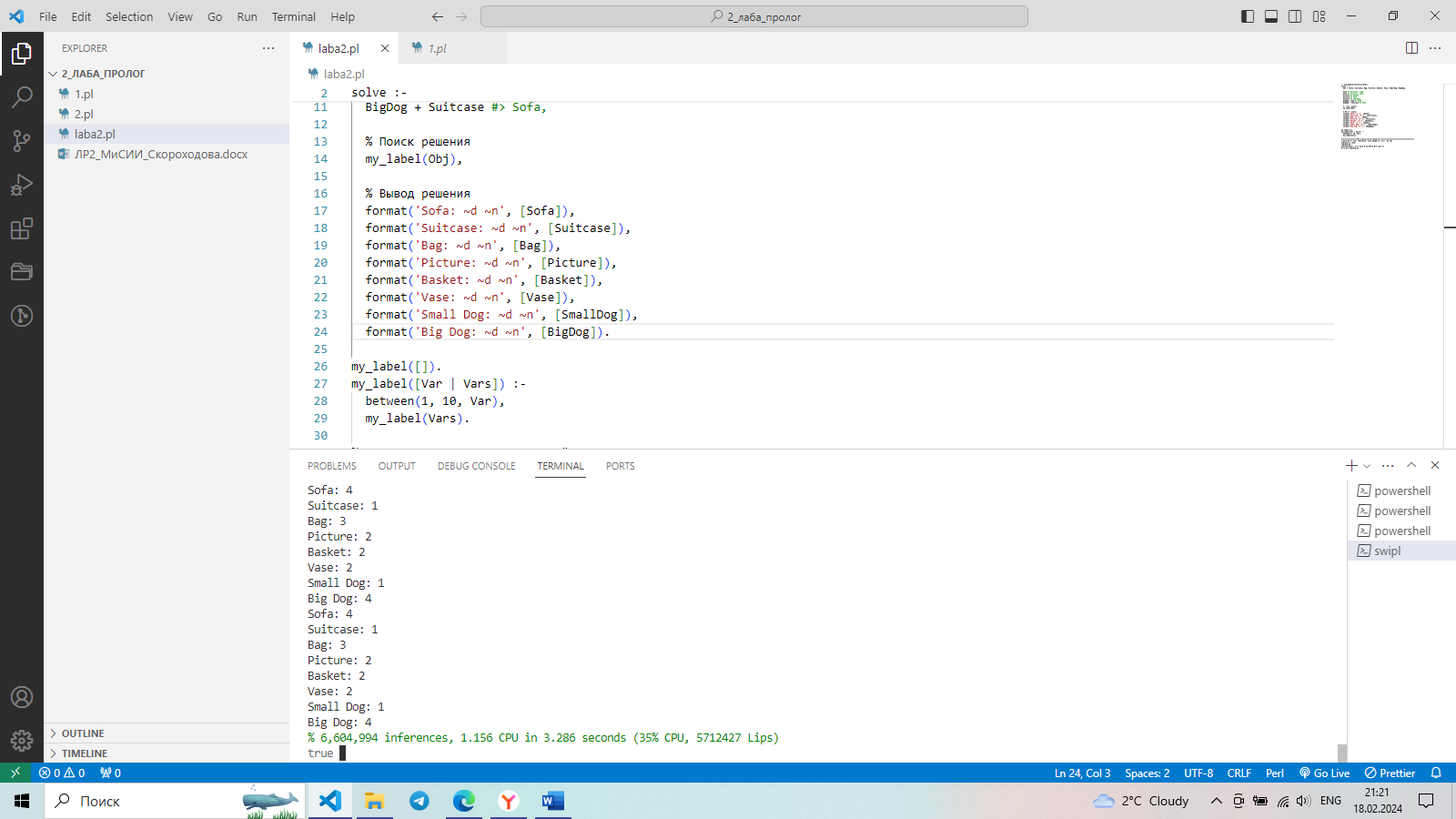


Рисунок 2 – time(rep(solve(),1000)).

5. Выводы

В ходе выполнения лабораторной работы было проведено изучение особенностей задач удовлетворения ограничений (CSP - Constraint Satisfaction Problem) и исследование основных методов поиска их решений средствами языка Пролог.

**Контрольные вопросы**:

**1. Приведите формальное определение задачи удовлетворения ограничений.**

Любая задача CSP определяется совокупностью трех составляющих:

1) множеством переменных X1, X2,…,Xn;

2) областью определения каждой переменной D1, D2,…,Dn;

3) множеством ограничений (отношений) С1, С2,…,Сm, каждое из которых включает некоторое подмножество переменных и задает допустимые комбинации значений для этого подмножества.

**2. Что понимают под состоянием SCP задачи, совместимым присваиванием, полным присваиванием, решением?**

*Состояние* задачи определяется путем присваивания значений некоторым или всем переменным. Присваивание, которое не нарушает никаких ограничений, называется *совместимым*. *Полным* называется присваивание, в котором участвует каждая переменная, а *решением* задачи CSP является полное присваивание, которое удовлетворяет всем ограничениям.

**3. Приведите пример формулировки задачи раскрашивания карты.**

Необходимо раскрасить карту, используя только 3 цвета. Соседние регионы на карте не должны раскрашиваться одним и тем же цветом.

Чтобы определить эту задачу в виде задачи CSP в качестве переменных будем использовать буквенные обозначения регионов на карте: A, B, C, D, E. Областью определения каждой переменной является множество цветов Di={red, green, blue}. Ограничения требуют, чтобы соседние регионы имели разные цвета, например, допустимые комбинации цветов для регионов А и В: {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}. В общем случае ограничения этой задачи можно записать в виде: Color(Xi) /= Color(Xj), где Xi и Xj переменные, обозначающие соседние регионы. Решение задачи заключается в том, чтобы всем переменным (A, B, C, D, E ) присвоить совместимые значения.

**4. Что понимают под графом ограничения?**

Задачу CSP удобно представлять в виде графа ограничений, узлы которого представляют переменные задачи, а дуги - ограничения.

**5. Как классифицируются переменные CSP задач?**

Переменные задачи могут быть ***дискретными или непрерывными***, с ***конечными или бесконечными областями определений***. Ниже рассматриваются только простые CSP задачи с дискретными переменными, характеризуемыми конечной областью определения. Если максимальный размер области определения равен d, то в худшем случае количество возможных полных присваиваний оценивается величиной O(). Поэтому время решения задачи экспоненциально зависит от количества переменных. Однако в большинстве практических случаев алгоритмы CSP общего назначения позволяют решать задачи на несколько порядков более крупные, например, по сравнению алгоритмами поиска решений в пространстве состояний.

**6. Как классифицируются ограничения CSP задач?**

***Ограничения*** (отношения) могут быть **унарными, бинарными и ограничениями высокого порядка** в зависимости от количества переменных. ***Унарные*** ограничения ограничивают значение одной переменной. Унарное ограничение можно устранить, удалив соответствующее значение из области определения переменной, нарушающее это ограничение. ***Бинарное*** ограничение связывает между собой две переменные. CSP задача, в которой используются только бинарные ограничения, может быть представлена графом ограничений. В ограничениях ***высокого порядка*** участвуют три и больше переменных. Ограничение высокого порядка может сведено к бинарным ограничениям путем введения вспомогательных переменных.

**7. Объясните метод «генерируй и тестируй».**

Метод также называют методом «образуй и проверь». В соответствии с этим методом **генерируются все возможные полные присваивания переменным, и каждое из них тестируется (проверяется) на совместимость**. Структура соответствующей программы весьма проста и выглядит в виде вложенных циклов по каждой переменной. В самом внутреннем цикле выполняется проверка каждого ограничения. Если они все выполняются, то текущее полное присвоение — решение задачи.

В большинстве случае «слепая» генерация всех полных присваиваний весьма неэффективна, т.к. приводит к разрастанию дерева поиска.

**8. Объясните поиск с возвратами при решении CSP задач.**

По сути, это тот же метод «генерируй и тестируй», но организованный в виде поиска в глубину, в котором присваивается значение очередной переменной, проверяются ограничения и выполняется возврат, если присвоение не допустимо.

Таким образом, проверка ограничений как бы погружается в процесс генерации решения, что позволяет ограничить разрастание дерева поиска.

На каждом шаге коэффициент ветвления b=d, всего n шагов, поэтому получаем d в степени n листьев.

Эффективность поиска зависит: 1) от порядка выбора переменных; 2) от порядка выбора возможных значений; 3) от возможности раннего обнаружения неудачи. Упорядочение выбора переменных и значений выполняют с помощью эвристик: 1) Наименьшего кол-ва оставшихся значений (MRV ― Minimum Remaining Values); 2) Степенной эвристики; 3) Наименее ограничительного значения.

***Степенная эвристика*** позволяет *уменьшить степень ветвления* за счет выбора *переменной*, которая участвует в *наибольшем количестве ограничений*. **MRV-эвристика** предусматривает выбор переменной с *наименьшим количеством оставшихся допустимых значений*. Такая переменная с наибольшей вероятностью вскоре приведет к неудаче, усекая тем самым дерево поиска. Степенная эвристика обычно используется для начального выбора переменных, а MRVэвристика в ходе дальнейших присваиваний.

В процессе поиска с возвратами множество переменных, которым уже присвоены значения, на каждом шаге расширяется на очередную переменную, если выполняются соответствующие ограничения. Процесс завершается, если получено полное присваивание.

**9. Объясните метод предварительной (опережающей) проверки.**

В случае поиска с возвратами ограничения, в которых участвует некоторая переменная, учитываются непосредственно в момент, когда происходит назначение значения этой переменной. Но ***выполняя опережающую проверку некоторых ограничений на предшествующих этапах поиска, можно резко сократить пространство поиска.***

В соответствии с методом предварительной проверки в момент присваивания значения переменной X просматривается каждая переменная Y, которой не присвоены значения и которая связана с X некоторым ограничением. При этом из области определения Y удаляется любое значение, которое несовместимо со значением, присвоенным переменной X.

**10. Объясните метод распространения ограничения.**

***Распространение ограничения*** — это общее название методов **обнаружения потенциальных несовместимостей на ранних этапах решения** задачи за счет распространения **последствий применения некоторого ограничения к одной из переменных.**

Для быстрого распространения ограничений выполняется проверка совместимости дуг. Дуга (X, Y ) называется совместимой, если для каждого значения х из области определения переменной X существует некоторое значение y из области определения переменной Y, которое удовлетворяет бинарному ограничению между этими переменными.

**11. В чем суть ограничения Alldiff ?**

Встречаются некоторые типы ограничений, которые могут обрабатываться более эффективно с помощью специальных алгоритмов, а не общих алгоритмов, рассмотренных выше. Например, ограничение Alldiff, которое указывает, что ***все переменные задачи должны иметь разные значения***. Один из простых способов проверки несовместимости для ограничения Alldiff сводится к правилу: ***если в ограничении участвуют m переменных и все они вместе взятые имеют n возможных значений, то при m>n ограничение не может быть удовлетворено.***

**12. Как на Прологе реализуется прямой метод «генерируй и тестируй»?**

На Прологе метод реализуется простой конъюнкцией двух целей, одна из которых выполняет генерацию возможных решений (полных присваиваний), а вторая проверяет, удовлетворяют ли эти решения ограничениям задачи:

**решить(X) :- генерировать\_решение(X), проверить\_ограничения(X).**

Если проверка завершается неудачей, то происходит возврат к цели генерировать\_решение, которая генерирует новое решение. Процесс продолжается до тех пор, пока не будет найдено решение, удовлетворяющее ограничениям, или генератор не исчерпает все альтернативные варианты предполагаемых решений.

**Какие предикаты Пролога обычно используют для генерации вариантов решений?**

В качестве основы генератора решений, часто используют запросы к базам данных или предикаты работы со списками: **принадлежит, удалить, перестановка.** Предикат принадлежит (встроенный предикат member) можно использовать не только для проверки того, что данный элемент входит в список, но и для того, чтобы получать (генерировать) элементы списка

**13. Приведите определения предикатов удалить и перестановка. Приведите примеры их использования.**

**удалить( X, [X|T], T ).**

**удалить( X, [Н|T], [Н|T1] ):-удалить(X, T, T1).**

**вставить(X,L1,L):-удалить(X,L,L1).**

**перестановка( [ ],[ ] ).**

**перестановка( [X|L], P ):-перестановка(L,L1), вставить(X, L1, P).**

**14. Объясните все примеры решений задачи раскрашивания карты на Прологе, приведенные в п. 2.2.3.**

**15. Объясните пример решения простейшей логической задачи на Прологе при выполнении целей в прямом (генерировать\_решение(X), проверить\_ограничения(X)) и обратном порядках.**

Эффективный вариант реализации можно получить и для примера простейшей логической задачи, рассмотренной выше. Оказывается, чтобы упорядочить список имен по возрасту методом поиска с возвратом и предварительной проверкой ограничений, ничего менять в основных определениях не надо. Достаточно просто сначала проверить выполнение ограничений, а затем дополнить полученный шаблон решения некоторыми не присвоенными значениями путем генерации перестановок. Иными словами, необходимо поменять местами цели в предикате решить1(X). Тогда получим следующее определение:

**решить2(X):- проверить\_ограничения(X), генерировать\_решение(X).**

Такое изменение порядка следования целей возможно, так как цель проверить\_ограничения(X) может не только проверять ограничения, но и выполнять генерацию частичных решений. В таком случае мы добиваемся того, что проверка ограничений максимально погружается в процесс генерации потенциальных решений.

**16. Как средствами Пролога определить основные статистические параметры выполнения программы?**