ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Как известно из молекулярной биологии, молекула белка синтезируется в рибосомах как последовательность аминокислотных остатков, которая затем спонтанно сворачивается в уникальную нативную пространственную структуру, за счет взаимодействия аминокислот друг с другом и с клеточным окружением. От формы этой структуры значительно зависят свойства белка, а ошибки в сворачивании могут вести к тяжелым последствиям для организма. Например, причиной многих аллергий является некорректное сворачивание некоторых белков, поскольку иммунная система не может производить антитела для определенных белковых структур [1].

Поскольку экспериментальное определение нативной структуры зачастую очень сложно и дорого, уже давно производятся попытки вычислительно предсказывать структуру белка, исходя из его начальной аминокислотной последовательности, которая обычно известна. Существуют различные подходы к этой задаче. Некоторые из них предполагают использование информации об известной трехмерной структуре белка, аминокислотная цепочка которого сходна с цепочкой рассматриваемого белка [2]. Другие используют только начальную полипептидную цепочку, и именно такой метод рассмотрен в данной работе.

Процесс сворачивания белка очень сложен, и механизмы, стоящие за ним, до конца не изучены. Поэтому для программного предсказания его структуры используются различные упрощенные модели. Они используют предположение, гласящее, что нативной является конформация белка, обладающая минимальной свободной энергией Гиббса [3], которая определяется взаимодействием аминокислотных остатков. Это могут быть весьма детализированные атомарные модели [4], или более простые, предполагающие размещение аминокислотных остатков на решетке в пространстве. Для последних, как правило, используется уменьшенный алфавит аминокислотных остатков. Это объясняется тем, что гидрофобные взаимодействия являются

основной движущей силой при фолдинге белка.

Примером такой модели является гидрофобно-полярная модель, используемая в данной работе. В ней существует два вида остатков: гидрофобные (H) и полярные (P). Цепочка располагается в пространстве в узлах кубической решетки (двумерной или трехмерной). Взаимодействующими считаются непоследовательные остатки, расположенные в соседних узлах. При этом взаимодействие двух гидрофобных остатков дает вклад в энергию, равный -1, а взаимодействие любых других ее не изменяет. Таким образом, задача заключается в том, чтобы найти такое расположение исходной цепочки на решетке, чтобы количество HH взаимодействий было максимальным. Пример такого расположения для последовательности `HHHPPHPHPHPPHPHPHPHPPH` можно видеть на рис. 1 (синим цветом выделены гидрофобные остатки, красным — полярные, зеленым — связи, дающие вклад в энергию). Энергия данной конфигурации равна -9.

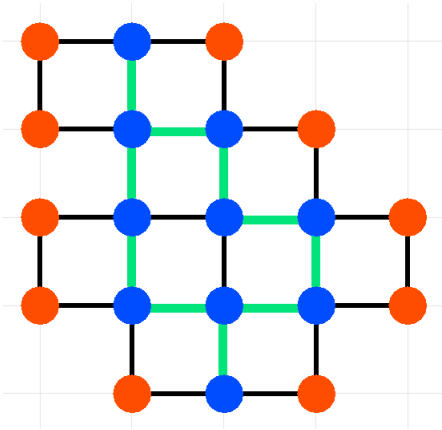


Рис. 1: Пример конфигурации гидрофобно-полярной модели

Несмотря на простоту модели, в [5] была доказана NP-полнота этой задачи. За годы ее существования было предложено множество способов ее решения, которые, тем не менее, в большинстве своем не дают точного результата. В данной работе предлагается еще один подход, заключающийся в сведении исходной задачи к задаче о выполнимости булевой формулы (SAT) и к использованию программирования в ограничениях (CSP). Выгода такого подхода заключается в том, что для решения этих, давно известных NP-полных задач существует множество высокоэффективных сторонних инструментов, которые активно развиваются. Кроме того, полученное таким образом решение всегда будет являться глобально оптимальным, в отличие от решений, предлагаемых в большинстве других подходов.

ГЛАВА 1. АНАЛИТИЧЕСКИЙ ОБЗОР

В данной главе приводится обзор существующих методов решения поставленной задачи, а также рассматриваются программные средства для решения задачи удовлетворения ограничений и задачи удовлетворимости булевых формул.

1.1 Существующие алгоритмы решения задачи

В настоящем разделе будут кратко изложены методы и алгоритмы решения задачи, описание которых было опубликовано в научно-исследовательских журналах, сборниках трудов к конференциям и другой литературе по данной теме.

1.1.1 Эволюционное моделирование

Для решения многих NP-полных задач нередко применяются методы, основанные на моделировании различных природных процессов, таких как естественный отбор или поведение насекомых. Описание примеров их применения к рассматриваемой задаче приведено в нижеследующих подразделах.

Использование клеточных автоматов

В работе [6] предлагается кодировать конфигурацию как серию поворотов относительно предыдущей позиции (пять вариантов для трехмерной решетки и три — для двумерной). Для сворачивания цепочки авторы вводят структуру под названием «нейронный клеточный автомат». Входные параметры искусственной нейронной сети определяются последствиями каждого из возможных движений текущего остатка в имеющейся на данный момент конфигурации. Для каждого из этих движений вычисляется вклад в общую

энергию (положительный или отрицательный). Если такое движение порождает коллизию, в качестве изменения энергии принимается очень большое, «штрафное» значение.

Затем для всех полученных движений вычисляется изменение в энергии для конфигурации, получаемой за несколько следующих шагов, где расположение для остатков выбирается «жадным» образом (вариант, осуществляющий наименьший вклад в текущую энергию). Все эти значения передаются на вход нейронной сети.

Выходной параметр сети определяет наиболее подходящий следующий поворот. Эта нейронная сеть применяется последовательно к каждому остатку в цепочке, и процесс повторяется в различных временных итерациях на всей последовательности аминокислотных остатков. Особи популяции для алгоритма дифференциальной эволюции кодируют в себе параметры нейронной сети — веса между ее слоями. В качестве фитнес-функции используется число HH-связей в итоговой конфигурации, полученной с помощью нейронной сети, соответствующей текущей особи. Причем для особей, порождающих конфигурации с самопересечениями, в качестве значения энергии используется число HH-связей до первой коллизии.

Особенностью данного подхода является итеративность построения конфигурации. Метод применим как для двумерных, так и для трехмерных решеток.

Использование муравьиных алгоритмов

Авторы [7] предлагают использовать для решения этой задачи муравьиные алгоритмы [8]. Задача представляется в виде графа, пути в нем — это решения. Метод предполагает использование множества агентов, которые ищут решения и отмечают наиболее перспективные из них с целью приблизиться к оптимальному за некоторое число итераций. Значения, хранящие информацию об успехе предыдущих решений, называются значениями феромона.

Для данной задачи решетка (двумерная или трехмерная) сама по себе представляет собой граф. При инициализации алгоритма всем ребрам присваивается исходное небольшое положительное значение феромона. Оно об-

новляется каждым агентом в процессе поиска решения по формуле

$$t_{ij} \to (1-\rho)t_{ij} + \rho\Delta t_0$$

а также после того, как все агенты завершили поиск, для ребер ij, принадлежащих наилучшему решению:

$$t_{ij} \to (1-\rho)t_{ij} + \rho\Delta t_{ij}$$

где Δt_{ij} — значение энергии для наилучшего решения, ρ — коэффициент испарения, позволяющий производить поиск в широкой окрестности текущего лучшего решения.

Выбор пути агентом на каждом шаге определяется матрицей значений феромона, представляющей информацию об имеющемся опыте поиска, и эвристической информацией, которую определяет получаемое изменение в энергии частичной конфигурации. Таким образом, вероятность выбора ребра ij:

$$P_{ij} = \frac{t_{ij}n_{ij}}{\sum\limits_{s \in allowed} t_{is}n_{is}}$$

для свободных позиций, и 0 — для недопустимых. Если текущий остаток полярный, его положение выбирается равномерным случайным образом среди свободных позиций. Если разместить остаток на данном шаге невозможно, алгоритм возвращается на несколько шагов назад.

Использование методов дифференциальной эволюции

В статье [9] авторы описывают применение методов дифференциальной эволюции для решения исходной задачи для двумерной решетки. Решение представляется в виде поворотов относительно предыдущего аминокислотного остатка. Поскольку данный метод предполагает использование вещественных векторов, положение остатка определяется попаданием веще-

ственного элемента x_i в один из интервалов, заданных константами $\alpha, \beta, \gamma, \delta$:

$$\begin{cases} \alpha < x_i \leq \beta \rightarrow \text{налево} \\ \beta < x_i < \delta \rightarrow \text{прямо} \\ \delta \leq x_i < \gamma \rightarrow \text{направо} \end{cases}$$

Решения с самопересечениями отбрасываются. В качестве начальной популяции принимается исходная цепочка (все остатки следуют один за другим).

В качестве функции приспособленности используется число *HH*-контактов непоследовательных остатков в конфигурации, соответствующей особи. Алгоритм параметризуется вероятностью кроссовера CR и весовым фактором F. Изменению могут подвергаться случайно выбранные вектора (что обеспечивает разнообразие в поколении), либо лучшие (что ускоряет сходимость). Для каждого выбранного таким образом вектора случайным образом выбираются вектора a, b, c. С вероятностью CR вектор x_i заменяется в следующем поколении на вектор

$$y_i = a_i + F(b_i - c_i)$$

если значение функции приспособленности для него больше, чем для исходного вектора.

Авторы предлагают комбинировать две вышеописанные стратегии выбора изменяемых векторов для достижения наилучших результатов: менять стратегию на случайную в случае отсутствия изменений функции приспособленности на протяжении многих поколений, и возвращаться к выбору лучших для ускорения сходимости. Критерием останова является произведенное число итераций.

Использование метода внешней оптимизации

Внешняя оптимизация, как и генетические алгоритмы, базируется на принципах естественного отбора, но сосредотачивается на удалении нежелательных локально оптимальных решений, заменяя их на стохастически сгенерированные.

Авторы [10] предлагают использовать две функции приспособленно-

сти: глобальную, которая, как и ранее, равна числу *HH*-контактов в конфигурации, и локальную для каждого остатка в цепочке.

На значение локальной функции приспособленности оказывают влияние три фактора. Первый из них — число свободных узлов решетки рядом с остатком. Таким образом принимается во внимание компактность расположения, свойственная оптимальным конфигурациям. Второй — число *HH*-связей для остатков, расположенный в соседних с текущим узлах (характеризует вклад гидрофобных остатков в энергию). Третий — число *HP*-пар, в которых P расположен от центра конфигурации дальше, чем *H* (характеризует «вытеснение» полярных остатков на внешний слой, вокруг гидрофобного ядра).

На первом шаге алгоритма в качестве начальной принимается любая допустимая конфигурация. Для каждого из остатков вычисляется значение локальной функции приспособленности. С вероятностью, обратно пропорциональной этому значению, выбирается остаток j, который будет мутирован.

Исходя из позиции j с помощью алгоритма *PERM*, описанного в [11], генерируется некоторое множество «соседних» конформаций, из которых выбирается одна с использование принципа колеса рулетки (вероятность выбора пропорциональна значению функции приспособленности).

Вышеописанные шаги повторяются, пока не будет выполнен критерий останова (например, достигнуто заданное число итераций).

Использование генетических алгоритмов

Оптимизация с помощью генетических алгоритмов является очень популярным методом решения данной задачи, и существует много вариантов ее применения. Некоторые из них описаны далее.

В работе [12] предлагается вариант использования генетических алгоритмов для решения данной задачи. В качестве особей используется вектор длины $N - 1$, где N — длина последовательности аминокислотных остатков. Каждый элемент этого вектора представляет собой положение текущего остатка относительно предыдущего (шесть позиций для трехмерной решетки и четыре — для двумерной).

Значение фитнес-функции для особи представляет собой число *HH*-контактов несмежных остатков в конфигурации. Отбор особей, формирую-

щих следующее поколение производится турнирным способом (из случайно выбранной пары особей выигрывает та, чье значение функции приспособленности выше).

К особям, полученным в результате отбора, применяется точечный кроссовер и мутация. Кроссовер осуществляется для пары особей путем взаимообмена позициями векторов от случайно выбранного k до N. В случае, если такой обмен порождает коллизию, производится попытка восстановить последовательность путем поиска подходящей свободной позиции для конфликтующих остатков. Если коллизий больше трех или их невозможно устранить, вектор возвращается в исходное состояние.

Мутация осуществляется путем замены случайно выбранного элемента вектора на какой-то другой из возможных. Если полученное изменение порождает коллизию, попытка повторяется до тех пор, пока конфигурация не будет допустимой. Если значение функции приспособленности для новой конфигурации больше, чем для предыдущей, новая особь замещает прежнюю.

Итерации отбора, скрещивания и мутации повторяются до тех пор, пока значение фитнес-функции не стабилизируется на протяжении десяти поколений.

Другим примером использования генетических алгоритмов является работа [13]. Ее авторы предлагают использовать для устранения коллизий в сгенерированных конформациях метод поиска с возвратом. В отличие от предыдущего алгоритма, если попытка найти свободный узел для конфликтующих остатков оказывается неуспешной, алгоритм возвращается на еще один шаг назад и осуществляет поиск уже с этой позиции, и так до тех пор, пока подходящее расположение не будет найдено для всей цепочки.

Еще одним отличием является использование принципа рулетки для отбора особей нового поколения. Авторы утверждают, что их подход позволяет достичь лучших результатов за меньшее число итераций, по сравнению с вышеописанным вариантом применения генетических алгоритмов.

1.1.2 Эвристические методы

В данном разделе описаны некоторые из эвристических методов, ориентированных на специфику задачи и имеющих, как правило, вероятностную

природу.

Метод контактного взаимодействия

Основная идея метода, предлагаемого авторами [14] в том, чтобы учесть тот факт, что, при формировании *HH*-контактов в конформации, подвижность аминокислотных остатков, находящихся в цикле, образованном двумя гидрофобными остатками, уменьшается. Первые два остатка фиксируются, а для остальных (с третьего по последний) вводится величина $f(i)$, характеризующая подвижность остатка i (то есть вероятность изменения его позиции по отношению к первым двум). Для остатков, чье движение ничем не ограничено, $f(i) = 0$. Для остатков, входящих в *HH*-цикл, состоящий из k остатков, $f(i) = -g(k)$. При этом циклы имеют аддитивное влияние на $f(i)$, то есть для остатка i, входящего одновременно в цикл длиной k и цикл длиной m, $f(i) = -g(k) - g(m)$. В статье описано исследование для $g(k) = 1$ и $g(k) = \frac{C}{A+k}$, где C и A — константы.

В качестве начальной структуры принимается развернутая цепочка. В каждой итерации алгоритма случайным образом выбирается остаток и принимается решение о том, следует ли его двигать, по критерию

$$Rand < exp\left(\frac{f(i)}{C_k}\right)$$

где $Rand$ — сгенерированное случайное число, C_k — параметр алгоритма. Если критерий удовлетворен, для остатка случайным образом выбирается новая позиция. Если конфигурация с такой позицией не является валидной, алгоритм возвращается в начало итерации. В обратном случае для новой конформации вычисляется энергия, обновляются значения $f(i)$, и алгоритм переходит в следующую итерацию. Критерием останова является число выполненных итераций.

Построение гидрофобного ядра с помощью ограничений

Алгоритм, предлагаемый авторами [15], базируется на дискретной геометрии и ориентирован на построение конформации с минимальной площадью поверхности гидрофобного ядра (именно такими и являются оптимальные конфигурации). Первым приближением к форме такого ядра является

сфера, заполненная гидрофобными остатками последовательности (как если бы не существовало ограничения связности для конформации). Кроме того, единичные полярные остатки должны находиться на поверхности ядра, иначе они порождали бы ядро с большей площадью поверхности. Другим соображением, используемым авторами метода, является тот факт, что в центре ядра должны находиться остатки из длинных гидрофобных подпоследовательностей: в обратном случае полярные остатки оказывались бы внутри ядра.

Формализуя вышеизложенные особенности, авторы накладывают геометрические ограничения, с помощью которых отсекают при укладке последовательности те ветви дерева поиска, которые не могут вести к формированию конформации с оптимальным ядром. Если не удалось уложить цепочку для некоторого ядра, в его геометрию вносятся изменения, и алгоритм повторяется заново.

Hydrophobic zippers

В основе метода, предлагаемого авторами [16], лежит предположение, гласящее, что гидрофобные остатки при фолдинге цепочки формируют пары, подобные зубцам застежки-«молнии». При формировании двумя пространственно близкими остатками первой HH-связи образуется ограничение, в результате которого сближаются другие два остатки, формируя вторую пару, и так далее (см. рис. 1.1).

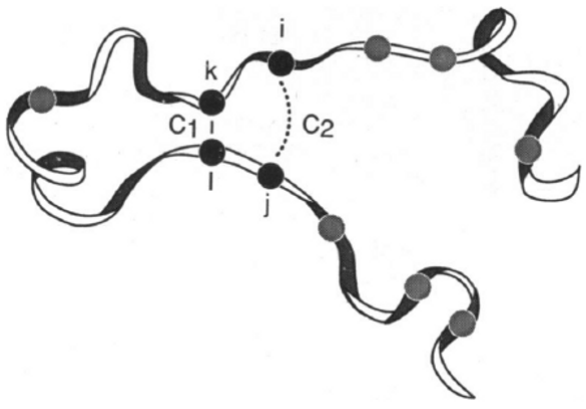


Рис. 1.1: Принцип работы HZ-метода [16]

Core-directed chain growth

Описываемый в статье [17] метод считается одним из самых эффективных, хотя и не гарантирует получение глобально оптимальной конфигурации. Авторы использовали особенность, заключающуюся в том, что конфигурации с минимальной энергией часто имеют своего рода «ядро», в котором сосредоточены гидрофобные остатки, в то время как полярные оказываются как бы в наружной оболочке. На каждом шаге построения конформации размещается некоторый сегмент последовательности, определенный нижеописанным образом.

На первом шаге алгоритма производится оценка размера ядра. Оно должно быть как можно ближе к квадрату, таким образом, чтобы все имеющиеся гидрофобные остатки могли в нем уместиться (например, для цепочки с 9 гидрофобными остатками ядро будет квадратом три на три). Пространство вокруг ядра и внутри него организовано оболочками или слоями.

Наращивание цепочки начинается со случайного выбора гидрофобного остатка i, такого, что остаток $i+1$ или $i-1$ также является гидрофобным. Этот остаток размещается на внешнем слое ядра. После этого начинается выбор и размещение сегментов последовательности от фиксированной таким образом позиции. Конформация может наращиваться в любую сторону, направление определяется случайным образом, пока не будет достигнут конец последовательности.

Длина выбираемого сегмента лежит в пределах от l_{min} до l_{max}, таким образом, чтобы в последней позиции был следующий гидрофобный остаток (если он не достижим, длина принимается за l_{max}). Далее осуществляется перебор всех возможных допустимых положений выбранного сегмента относительно уже сформированной частичной конформации. Для каждого из них вычисляется значение эвристической функции. Эта функция направлена на максимизацию числа HH-контактов и гидрофобных остатков в ядре, при вытеснении полярных остатков в наружный слой и уменьшении числа HP-контактов. Среди конфигураций сегмента с одинаковым значением эвристической функции одна выбирается случайно и равномерно, и алгоритм переходит к следующему сегменту. Шаги повторяются до тех пор, пока вся цепочка не будет размещена.

Алгоритм повторяется несколько раз, и для небольшого числа наилуч-

ших конфигураций производится «тонкая настройка»: некоторое случайное число аминокислотных остатков с конца цепочки удаляется и укладывается заново вышеописанным алгоритмом.

Критерием останова является время работы.

Использование алгоритма PERM

Pruned-enriched-Rosenbluth method [11], изначально созданный для построения цепочек полимеров, был успешно применен для решения рассматриваемой задачи. Его идея состоит в том, чтобы по мере построения хранить и обновлять некоторое множество конфигураций, каждой из которых сопоставлен вес W_n. При этом, если на очередном шаге предполагаемый вес новой конфигурации W_n^{pred} меньше порогового значения $W_n^<$, конфигурация отсекается с вероятностью 1/2, а если он превосходит пороговое значение $W_n^>$, конфигурация заменяется k копиями, каждой из которых присваивается вес W_n^j. При этом пороговые значения могут изменяться в процессе выполнения алгоритма, не влияя на корректность результата. Но от этих значений зависит эффективность алгоритма.

Число копий k определяется как

$$k = min\{k_{free}, \lceil \frac{W_n^{pred}}{W_n^>} \rceil\}$$

где k_{free} — число свободных позиций, куда можно поместить текущий остаток. При этом размещение остатка n в свободный узел a осуществляется с вероятностью

$$p_n^a = \frac{(k_{free}^a + \frac{1}{2})exp(-\frac{\Delta E_a}{T})}{\sum\limits_{a \in A}(k_{free}^a + \frac{1}{2})exp(-\frac{\Delta E_a}{T})}$$

где k_{free}^a — число свободных позиций при попытке разместить остаток n в позицию a, A — множество свободных позиций.

Вес последовательности обновляется как

$$W_n = W_{n-1}exp(-\frac{\Delta E_a}{T})$$

где $W_1 = 1$, ΔE_a — изменение энергии при размещении остатка n в позиции a, T — параметр алгоритма.

Предполагаемый вес конфигурации определяется как

$$W_n^{pred} = W_{n-1} exp\left(-\frac{\sum\limits_{a \in A} \Delta E_a}{k_{free} T}\right)$$

Пороговые значения весов определяются как

$$W_n^{>} = C\left(\frac{S_n}{C_n}\right)\left(\frac{C_n}{control}\right)^2$$

$$W_n^{<} = 0.2 W_n^{>}$$

где C — положительная константа, $\left(\frac{S_n}{C_n}\right)$ — средний вес конфигураций длины n, сгенерированных на данный момент, C_n — их число, $control$ — параметр алгоритма.

1.1.3 Точные методы

В рассмотренных статьях также было предложено несколько точных методов решения данной задачи, один из которых был реализован в данной работе и использован для сравнения.

Использование программирования в ограничениях

Авторы [18] предлагают вариант использования программирования в ограничениях для решения данной задачи. Конфигурация задается массивами целочисленных координат, на которые накладываются ограничения связности и отсутствия самопересечений. Также вводится двумерный массив переменных $Contact_{ij}$ — расположены ли остатки i и j в соседних узлах, выражающийся через координаты. Функция энергии выражается через эти переменные как

$$\sum_{i+1<j \bigwedge s_i=H \bigwedge s_j=H} -Contact_{ij}$$

где s_j — элемент последовательности с номером j. Этот метод был реализован и улучшен в рамках данной работы, и описан подробнее в разделе, посвященном теоретической части исследования.

Сведение к Weighted MAX-SAT

Авторы [19] предлагают в качестве решения сведение исходной задачи к другой NP-полной задаче, решаемой затем сторонним инструментом. Однако, в отличие от классической задачи о выполнимости булевой формулы, используемой в данной работе, они используют другую, известную как Weighted MAX-SAT. На вход ей подается булева формула, где каждому дизъюнкту присвоен некоторый вес. Требуется найти такой набор переменных, для которого суммарный вес удовлетворенных дизъюнктов был бы максимальным.

Конфигурация кодируется поворотами, по два бита на направление (01 — направо, 00 — вниз, 10 — налево, 11 — вверх). Преимущество такого кодирования заключается в том, что оно автоматически гарантирует связность последовательности. Энергетическая функция конфигурации q состоит из трех компонент:

$$E(q) = E_{back}(q) + E_{overlap}(q) + E_{pair}(q)$$

где $E_{pair}(q)$ отвечает за вклад в энергию при взаимодействии гидрофобных остатков, а $E_{back}(q)$ и $E_{overlap}(q)$ — функции, позволяющие избежать коллизий. Первая из них штрафует движения в направлении, противоположном движению на предыдущем шаге (то есть конфликт между остатками i и $i+1$), а вторая — любые последующие коллизии.

Также в работе предлагается альтернативный способ кодирования конфигураций, называемое «ромбовидным». Суть его заключается в кодировании позиции остатка i относительно первого. Иллюстрацию можно видеть на рис. 1.2. Число возможных позиций растет в каждом новом слое, по мере удаления от центра, и с ним увеличивается число бит, требуемых для кодирования расположения каждого последующего остатка. Поскольку такой способ кодирования не гарантирует связности последовательности, к энергетической функции добавляется еще две компоненты: $E_{one}(q)$, гарантирующая, что каждый остаток занимает ровно одну позицию из возможных, и $E_{connect}(q)$, отвечающая непосредственно за то, что последовательные остатки находятся в соседних узлах решетки.

Энергетические функции выражаются через двоичные переменные,

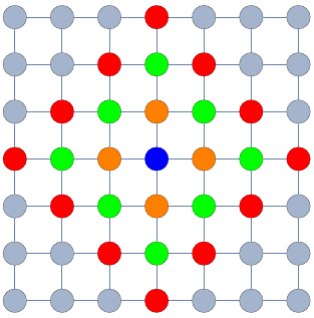


Рис. 1.2: Ромбовидное кодирование [19]

кодирующие конфигурацию, и различные коэффициенты (положительные или отрицательные, в зависимости от того, штрафная ли это компонента или нет), и задают веса для дизъюнктов, передаваемых на вход инструменту, решающему Weighted MAX-SAT.

В своей работе авторы также предлагают способ использования целочисленного линейного программирования для решения исходной задачи, заключающийся в преобразовании в формы Weighted MAX-SAT в данные, принимаемые на вход инструментами для решения задач целочисленного линейного программирования.

1.1.4 Прочие методы

В данном разделе рассмотрены еще несколько методов вероятностной природы, предлагаемых в литературе для решения этой задачи.

Перебор с вероятностными отсечениями

Идея решения, предложенного [20], заключается в том, чтобы вероятностным образом отсекать наименее перспективные ветви дерева решений при переборе конфигураций. Алгоритм параметризуется двумя параметрами, представляющими собой вероятности: p_1 и p_2. На каждом шаге алгоритма поддерживаются и обновляются две величины: U_k — наименьшее значение энергии, полученное для частичной конформации длины k за все время работы алгоритма, и Z_k — среднее значение энергии для частичной конформации длины k. Для каждой из возможных новых позиций остатка k вычисляется значение энергии E_k. Если текущий остаток полярный, ветка решения сохраняется, и алгоритм рекурсивно уходит в следующую итерацию. Если остаток

гидрофобный и $E_k < U_k$, ветка решения также сохраняется, если $E_k > Z_k$ — решение отсекается с вероятностью p_1, если $E_k < Z_k$ и $E_k > U_k$ — решение отсекается с вероятностью p_2.

Использование метода Монте-Карло

Метод Монте-Карло, описанный в [21], был одним из первых способов, предложенных для решения этой задачи. В силу того, что он является очень общим, конфигурация может быть закодирована как угодно (в абсолютных или относительных координатах). Выполнение алгоритма начинается со случайно сгенерированной конфигурации. Затем из текущей конфигурации S_1 путем внесения случайного изменения генерируется конфигурация S_2. Если энергия вновь полученной конфигурации меньше, чем энергия S_1, она принимается за текущую, и алгоритм переходит в следующую итерацию. В обратном случае S_2 принимается по критерию

$$Rand < exp\left(\frac{E_1 - E_2}{c_k}\right)$$

где E_1 и E_2 — энергии соответствующих конфигураций, c_k — параметр, уменьшающийся в процессе итераций для обеспечения сходимости.

Эволюционный алгоритм Монте-Карло

Авторы [22] предлагают использовать для решения задачи эволюционный алгоритм Монте-Карло. Конформация кодируется в форме целочисленного вектора z со значениями из множества $0, 1, 2$, задающими положение остатка относительно предыдущего: справа, слева или впереди. Энергетическая функции конфигурации выражается через $H(z)$ и распределение Больцмана как

$$f_i(z) \propto exp\left(-\frac{H(z)}{\tau}\right)$$

где τ называется температурой системы.

Изначально вводится последовательность распределений $f_1, ..., f_N$, где

$$f_i(z) \propto \left(-\frac{H(z)}{t_i}\right)$$

Последовательность температур $\mathbf{t} = (t_1, ..., t_N)$ формирует «лестницу» с $t_1 < ... < t_N \equiv \tau$. Величина $\mathbf{t}$, как и N — размер популяции, являются параметрами алгоритма.

Пусть z_i выбран из f_i, тогда $\mathbf{z} = \{z_1, ..., z_N\}$ формируют популяцию. Распределение Больцмана для популяции выглядит как

$$f(\mathbf{z}) \propto \left(-\sum_{i=1}^{N} \frac{H(z_i)}{t_i} \right)$$

К популяции $\mathbf{z}$ применяется оператор мутации, в результате чего формируется популяция $\mathbf{z}'$, отличающаяся от $\mathbf{z}$ элементом z_m . Вектор $\mathbf{z}$ заменяется на $\mathbf{z}'$ с вероятностью $min(1, r_m)$ по правилу Метрополиса-Гастингса:

$$r_m = \frac{f(\mathbf{z}')}{f(\mathbf{z})} \frac{T(\mathbf{z}|\mathbf{z}')}{T(\mathbf{z}'|\mathbf{z})} = exp\left(-(H(z'_m) - H(z_m))/t_m\right) \frac{T(\mathbf{z}|\mathbf{z}')}{T(\mathbf{z}'|\mathbf{z})}$$

где $T(.|.)$ представляет собой вероятность перехода между двумя популяциями (вид этой функции зависит от характера мутации).

Операция скрещивания производится над двумя случайно выбранными из $\mathbf{z}$ родительскими элементами z_a и z_b путем их разбиения в позиции c и обмена частей последовательности, лежащих справа от этой точки. Новая популяция формируется заменой родительских элементов наследниками с лучшим значением энергии. Вероятность замены прежней популяции на новую $min(1, r_c)$:

$$r_c = \frac{f(\mathbf{z}')}{f(\mathbf{z})} \frac{T(\mathbf{z}|\mathbf{z}')}{T(\mathbf{z}'|\mathbf{z})} = exp\left(-(H(z'_a) - H(z_a))/t_a - (H(z'_b) - H(z_b))/t_b\right) \frac{T(\mathbf{z}|\mathbf{z}')}{T(\mathbf{z}'|\mathbf{z})}$$

где $T(\mathbf{z}'|\mathbf{z}) = P((z_a, z_b)|\mathbf{z})P((z'_a, z'_b)|(z_a, z_b))$, $P((z_a, z_b)|\mathbf{z})$ означает вероятность выбора пары (z_a, z_b) из популяции $\mathbf{z}$, $P((z'_a, z'_b)|(z_a, z_b))$ означает вероятность генерации (z'_a, z'_b) из родительских элементов (z_a, z_b).

Родительские элементы выбираются методом колеса рулетки с больцмановскими весами

$$w(z) = \frac{1}{C(\mathbf{z})} exp(-H(z)/t_s)$$

где $C(\mathbf{z}) = \sum_{i=1}^{N} exp(-H(z_i)/t_N)$.

Операция обмена элементов преобразует популяцию
$\mathbf{z} = \{z_1, t_1, ..., z_i, t_i, ..., z_j, t_j, ..., z_N\}$ в популяцию
$\mathbf{z}' = \{z_1, t_1, ..., z_j, t_i, ..., z_i, t_j, ..., z_N\}$. Новая популяция принимается с вероятностью $min(1, r_e)$:

$$r_e = \frac{f(\mathbf{z}')}{f(\mathbf{z})} \frac{T(\mathbf{z}|\mathbf{z}')}{T(\mathbf{z}'|\mathbf{z})} = exp\left((H(z_i) - H(z_j)) \left(\frac{1}{t_i} - \frac{1}{t_j} \right) \right) \frac{T(\mathbf{z}|\mathbf{z}')}{T(\mathbf{z}'|\mathbf{z})}$$

В процессе итерации алгоритма к популяции применяется или оператор мутации, или оператор скрещивания с вероятностями p_m и $1 - p_m$ соответственно (p_m — параметр алгоритма). Затем производится попытка обменять z_i и z_j для $N - 1$, выбранных равномерно из $\{1..N\}$ при $j = i + 1$ или $j = i - 1$ с вероятностью 0.5.

1.2 Программные инструменты для решения NP-полных задач

Существует ряд NP-полных задач, поиску решения которых посвящено очень много исследований. Среди них — задачи удовлетворимости булевой формулы, задача удовлетворения ограничений, задача раскраски графа. В результате этих исследований было разработано множество высокопроизводительных программных средств (солверов), эффективность работы которых продолжает расти с каждым годом.

Существует немало случаев использования возможностей этих инструментов для решения других NP-полных задач, путем сведения последних к задаче, решаемой солвером. В качестве примера можно привести метод идентификации детерминированных конечных автоматов, описанный в [23], либо способы построения детерминированных конечных автоматов по сценариям работы, изложенные в работах [24], [25], [26].

1.2.1 Инструменты для решения задачи удовлетворимости булевой формулы

Задача удовлетворимости булевой формулы заключается в том, чтобы для заданной булевой формулы $F(x_1, ..., x_n)$, в которой используются только операторы $\{\bigwedge, \bigvee, \neg, (,)\}$, определить, существует ли такой набор $a_1, ..., a_n, a_i \in \{true, false\}$, чтобы при их подстановке формула принимала значение «истина»: $F(a_1, ..., a_n) = true$.

Согласно теореме Кука, эта задача NP-полна [27], что означает, что для ее решения существуют только алгоритмы, обладающие экспоненциальной сложностью в худшем случае. Тем не менее, разработаны различные программные инструменты, весьма эффективно решающие эту задачу для десятков тысяч переменных и миллионов ограничений во многих случаях [28]. Ежегодно проводятся соревнования между этими продуктами, определяющие победителей в различных номинациях, таких как решение прикладных примеров, трудно вычислимых либо случайно сгенерированных, способность эффективно использовать параллельные вычисления или работать в одном потоке, и других [29].

В качестве универсального входного формата эти программные средства, как правило, используют язык *DIMACS*. Это простой текстовый, позволяющий описывать булевы формулы, заданные в конъюнктивной нормальной форме (КНФ). КНФ предполагает, что формула имеет вид $F(x_1, ..., x_n) = (x_1 \bigvee ... \bigvee x_t) \bigwedge ... \bigwedge (\neg x_k \bigvee ... \bigvee x_n)$, то есть состоит из скобок, соединенных оператором конъюнкции, в которые входят переменные и их отрицания, соединенные, в свою очередь, оператором дизъюнкции. В [27] показано, что любую булеву формулу можно привести к КНФ с полиномиальным увеличением длины.

Запись формулы на языке *DIMACS* состоит из заголовка, содержащего число переменных и число скобок (дизъюнктов) в формуле, и записи самих дизъюнктов в виде строк с номерами переменных, записанных через пробел. Отрицательные номера означают оператор отрицания. Каждая строка заканчивается нулем. В качестве примера рассмотрим формулу

$$F(a_1, a_2, a_3, a_4) = (a_1 \bigvee a_2) \bigwedge (\neg a_3 \bigvee a_4)$$

На языке *DIMACS* она выглядит как:

```
p cnf 4 2
1 2 0
-3 4 0
```

В качестве результата возвращается список удовлетворяющих формулу переменных, оканчивающийся на ноль, либо утверждение, что она неудовлетворима. Например, ответом на приведенный пример может быть строка:

```
1 -2 -3 -4 0
```

Победителями последних соревнований являются такие продукты как *Lingeling*, *Plingeling* и *Treengeling* [30], а также *Glucose* [31].

1.2.2 Инструменты для решения задачи удовлетворения ограничений

Формальная постановка задачи выглядит следующим образом. Дано множество переменных $X = \{X_1, ...X_n\}$, для каждой из которых задан непустой домен D_i, и множество ограничений $C = \{C_1, ..., C_m\}$. Требуется найти набор переменных, удовлетворяющих ограничениям.

Для записи входных данных инструментов, решающих задачу удовлетворения ограничений, также существует ряд форматов, одним из которых является язык *MiniZinc* [32].

Этот язык является гораздо более понятным для человека, чем язык *DIMACS*, и позволяет записывать непосредственно весьма сложные прикладные задачи. В список его возможностей входит использование целочисленных переменных, множеств и массивов, арифметических и логических операторов, условных выражений и предикатов. Кроме того, он позволяет тонко управлять процессом поиска: например, находить все существующие решения.

Для программных продуктов, поддерживающих язык *MiniZinc*, также существуют свои соревнования [32]. Среди победителей последнего из них инструмент *Opturion CPX* [33].

В работе [34] описаны и проанализированы различные способы взаимного сведения задач о выполнимости булевой формулы и выполнения ограничений. Таким образом, если некоторую задачу можно решить с использованием одного метода, то для нее подойдет и другой. Преимуществом программирования в ограничениях является, несомненно, простота и понятность синтаксиса, равно как и возможность отделить модель решения от входных данных. Использование инструментов для решения задачи выполнимости булевой формулы требует написания программы, которая по входным данных генерировала бы формулы, что само по себе не просто. Кроме того, поиск ошибок в таком коде и его поддержка оказываются нелегкой задачей. Но, с другой стороны, этот подход зачастую позволяет добиться высокой эффективности за счет более полного использования специфики задачи.

ГЛАВА 2. РАЗРАБОТАННЫЕ АЛГОРИТМЫ

В этой главе приведено описание предлагаемых подходов к решению поставленной задачи. Первый подход предполагает использование программирования в ограничениях, второй – сведение к задаче о выполнимости булевой формулы.

2.1 Формальная постановка задачи

Задача поиска оптимальной конфигурации гидрофобно-полярной модели протеина формулируется следующим образом. На вход подается последовательность длины N, состоящая из H и P. В качестве результата должно возвращаться описание пространственного расположения цепочки на кубической (или квадратной, в двумерном случае) решетке такое, что число непоследовательных H остатков, находящихся в соседних (отличающихся на единицу по одной из координат) узлах было максимальным. При этом в полученной конфигурации цепочка должна оставаться связной и не иметь самопересечений.

Методы, используемые в данной работе, имеют некоторые особенности. А именно, решением задач, к которым сводится исходная, является ответ «да» либо «нет». Поэтому, чтобы использовать возможности этих методов, предлагается решать сначала параметризованный вариант задачи. Он формулируется следующим образом: «Найти пространственные координаты аминокислотных остатков заданной последовательности, свернутой на решетке заданного размера таким образом, чтобы модуль суммарной энергии конфигурации был не меньше заданного числа, если такая конфигурация существует». Алгоритм, решающий эту задачу, принимает на вход следующие величины.

- Массив R длины N, R, где $R[i] = true$ если остаток i гидрофобный,

$R[i] = true$ если остаток i полярный для $i \in [1..N]$, N – длина исходной последовательности.

- Число L, ограничивающее размер решетки ($L \times L$ узлов для двумерной и $L \times L \times L$ для трехмерной).
- Число K, ограничивающее снизу значение суммарной энергии конфигурации.

Результатом работы являются одномерные массивы целочисленных координат x и y (и z для трехмерного случая), где $x[i] \in [0..L], y[i] \in [0..L], z[i] \in [0..L], i \in [1..N]$, либо утверждение, что такой конфигурации не существует.

2.2 Решение с использованием программирования в ограничениях

В настоящем разделе предлагается описание двух способов решения задачи программированием в ограничениях, отличающихся способом получения требуемого значения энергии.

2.2.1 Переменные

И для первого, и для второго способа вводится двумерный массив $neighbors$, $neighbors[i][j] \in [0..1], i \in [1..N], j \in [1..N]$ — являются ли i, j соседями (лежат в смежных узлах).

Для второго способа также потребуются два целочисленных массива $begin$ и end длины K. $begin[i]$ содержит порядковый индекс начального остатка в HH-паре с номером i, а $end[i]$ – порядковый индекс конечного остатка.

2.2.2 Ограничения

На вышеописанные переменные накладываются следующие общие ограничения:

- Определение «соседей»: заданные остатки находятся в соседних узлах

$$\begin{aligned} neighbors[i][j] = 1 \leftrightarrow \\ ((x[j] = x[i] \bigwedge y[j] = y[i] + 1 \bigwedge z[j] = z[i]) \bigvee \\ (x[j] = x[i] \bigwedge y[j] = y[i] - 1 \bigwedge z[j] = z[i]) \bigvee \\ (x[j] = x[i] + 1 \bigwedge y[j] = y[i] \bigwedge z[j] = z[i]) \bigvee \\ (x[j] = x[i] - 1 \bigwedge y[j] = y[i] \bigwedge z[j] = z[i]) \bigvee \\ (x[j] = x[i] \bigwedge y[j] = y[i] \bigwedge z[j] = z[i] + 1) \bigvee \\ (x[j] = x[i] \bigwedge y[j] = y[i] \bigwedge z[j] = z[i] - 1)) \\ i < j \bigwedge((j - i) \bmod 2 = 1), i \in [0..N], j \in [0..N] \end{aligned} \tag{2.1}$$

- Поскольку выбор позиций двух первых остатков не влияет на результат, можно их зафиксировать, сократив тем самым число шагов, требуемых для решения задачи:

$$x[0] = 0 \bigwedge y[0] = 0 \bigwedge z[0] = 0 \bigwedge x[1] = 1 \bigwedge y[1] = 0 \bigwedge z[1] = 0 \tag{2.2}$$

- Никакие два остатка не расположены в одном и том же узле

$$x[i] = x[j] \rightarrow y[i] \neq y[j] \bigvee z[i] \neq z[j] \forall i < j, i \in [0..N], j \in [0..N] \tag{2.3}$$

- Последовательность связна (любые два последовательных остатка являются соседями)

$$neighbors[i][i + 1] = 1 \forall i \in [0..N - 1] \tag{2.4}$$

Чтобы значение функции энергии было равно K должно существовать K различных пар соседей, в которых оба остатка гидрофобны (HH-пар). Для задания этого ограничения предлагается два способа:

Способ 1 Потребуем, чтобы число пар, являющихся «соседями», где оба остатка гидрофобны, было не меньше K, и исключим из рассмотре-

ния последовательные остатки, поскольку их вклад в общую энергию не зависит от конфигурации:

$$R[i] = true \bigwedge R[j] = true \bigwedge neighbors[i][j] \forall i < j - 2$$

Способ 2 Из соображений симметрии, потребуем, чтобы начальный индекс в паре всегда был меньше конечного, исключив последовательные остатки :

$$begin[i] + 2 < end[i] \forall i \tag{2.5}$$

Кроме того, чтобы избежать неоднозначности, необходимо, чтобы пары индексов $(begin[i], end[i])$ были упорядочены:

$$begin[i] \leq begin[j] \forall i < j \tag{2.6}$$

Если для некоторых двух пар остатков начальные индексы совпадают, то конечные должны отличаться:

$$begin[i] = begin[j] \rightarrow end[i] < end[j] \forall i < j \tag{2.7}$$

Вышеописанные переменные и ограничения формирую входные данные для стороннего программного инструмента, который и возвращает результат, либо сообщает о том, что решения не существует.

2.3 Сведение поставленной задачи к SAT

Сведение к SAT производится путем записывания тех же ограничений через булевы переменные и формулы, с тем условием, чтобы формулы находились в конъюнктивной нормальной форме (КНФ). Последнее требование проистекает из общепринятого стандарта для входных данных, принимаемых на вход инструментами для решения SAT.

2.3.1 Переменные

Для всех нижеследующих переменных $i \in [1..N], j \in [1..N], a \in [0..L]$.

X_{ia} — верно ли, что $x[i] = a$, Y_{ia} — верно ли, что $y[i] = a$, Z_{ia} — верно ли, что $z[i] = a$. Значения этих переменных выражают решение задачи.

$Neighbor_{ij}$ — верно ли, что остатки i и j расположены в соседних узлах решетки.

$Right_{ij}$ — верно ли, что остаток j расположен справа от остатка i.

$Left_{ij}$ — верно ли, что остаток j расположен слева от остатка i.

Up_{ij} — верно ли, что остаток j расположен впереди остатка i.

$Down_{ij}$ — верно ли, что остаток j расположен сзади остатка i.

Top_{ij} — верно ли, что остаток j расположен сверху от остатка i.

$Bottom_{ij}$ — верно ли, что остаток j расположен снизу от остатка i.

H_{ij} — верно ли, что и остаток i, и остаток j являются гидрофобными.

$Begin_{ia}$ — верно ли, что в HH-паре i начальный индекс равен a.

End_{ia} — верно ли, что в HH-паре i конечный индекс равен a.

2.3.2 Ограничения

Ограничение (2.1) записывается следующим образом для $i, j \in [1...N]$:

$$Neighbor_{ij} \leftrightarrow Right_{ij} \bigvee Left_{ij} \bigvee Up_{ij} \bigvee Down_{ij}$$

$$Right_{ij} \leftrightarrow (\bigvee_{a\in[0..L]} X_{ia} \bigwedge X_{ia+1}) \bigwedge (\bigvee_{a\in[0..L]} Y_{ia} \bigwedge Y_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Z_{ia} \bigwedge Z_{ia})$$

$$Left_{ij} \leftrightarrow (\bigvee_{a\in[0..L]} X_{ia} \bigwedge X_{ia-1}) \bigwedge (\bigvee_{a\in[0..L]} Y_{ia} \bigwedge Y_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Z_{ia} \bigwedge Z_{ia})$$

$$Up_{ij} \leftrightarrow (\bigvee_{a\in[0..L]} X_{ia} \bigwedge X_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Y_{ia} \bigwedge Y_{ia-1}) \bigwedge (\bigvee_{a\in[0..L]} Z_{ia} \bigwedge Z_{ia})$$

$$Down_{ij} \leftrightarrow (\bigvee_{a\in[0..L]} X_{ia} \bigwedge X_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Y_{ia} \bigwedge Y_{ia+1}) \bigwedge (\bigvee_{a\in[0..L]} Z_{ia} \bigwedge Z_{ia})$$

$$Top_{ij} \leftrightarrow (\bigvee_{a\in[0..L]} X_{ia} \bigwedge X_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Y_{ia} \bigwedge Y_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Z_{ia} \bigwedge Z_{ia-1})$$

$$Bottom_{ij} \leftrightarrow (\bigvee_{a\in[0..L]} X_{ia} \bigwedge X_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Y_{ia} \bigwedge Y_{ia}) \bigwedge (\bigvee_{a\in[0..L]} Z_{ia} \bigwedge Z_{ia+1})$$

Чтобы переписать эти формулы в КНФ, необходимо ввести дополнительные переменные (с индексами $i \in [1..N], j \in [1..N]$). Для записи равенства по всем координатам, для x используются:

$Match\text{X}_{ij}$ — верно ли, что $x[i] = a, x[j] = a \forall a \in [1..L]$

$Match\text{X+1}_{ij}$ — верно ли, что $x[i] = a, x[j] = a + 1 \forall a \in [1..L-1]$

$Match\text{X-1}_{ij}$ — верно ли, что $x[i] = a, x[j] = a - 1 \forall a \in [2..L]$

Для y:

$Match\text{Y}_{ij}$ — верно ли, что $y[i] = a, y[j] = a \forall a \in [1..L]$

$Match\text{Y+1}_{ij}$ — верно ли, что $y[i] = a, y[j] = a + 1 \forall a \in [1..L-1]$

$Match\text{Y-1}_{ij}$ — верно ли, что $y[i] = a, y[j] = a - 1 \forall a \in [2..L]$

Для z:

$Match\text{Z}_{ij}$ — верно ли, что $z[i] = a, z[j] = a \forall a \in [1..L]$

$Match\text{Z+1}_{ij}$ — верно ли, что $z[i] = a, z[j] = a + 1 \forall a \in [1..L-1]$

$Match\text{Z-1}_{ij}$ — верно ли, что $z[i] = a, z[j] = a - 1 \forall a \in [2..L]$

Но, поскольку выразить эти ограничения формулой полиномиальной длины в КНФ по-прежнему невозможно, для выражения равенства по фиксированной координате вводится еще одна группа переменных ($i \in [1..N], j \in [1..N]$). Для x:

$Match\text{X}_{ija}$ — верно ли, что $x[i] = a, x[j] = a$

$Match\text{X+1}_{ija}$ — верно ли, что $x[i] = a, x[j] = a + 1$

$Match\text{X-1}_{ija}$ — верно ли, что $x[i] = a, x[j] = a - 1$

Для y:

$Match\text{Y}_{ija}$ — верно ли, что $y[i] = a, y[j] = a.$

$Match\text{Y+1}_{ija}$ — верно ли, что $y[i] = a, y[j] = a + 1.$

$Match\text{Y-1}_{ija}$ — верно ли, что $y[i] = a, y[j] = a - 1.$

Для z:

$Match\text{Z}_{ija}$ — верно ли, что $z[i] = a, z[j] = a.$

$MatchZ{+}1_{ija}$ — верно ли, что $z[i] = a, z[j] = a + 1$.

$MatchZ\text{-}1_{ija}$ — верно ли, что $z[i] = a, z[j] = a - 1$.

Таким образом, преобразования для описания положения в соседних узлах будет выглядеть как :

$$Right_{ij} \leftrightarrow MatchZ_{ij} \bigwedge MatchY_{ij} \bigwedge MatchX{+}1_{ij}, i \in [1..N], j \in [1..N]$$

$$Left_{ij} \leftrightarrow MatchZ_{ij} \bigwedge MatchY_{ij} \bigwedge MatchY\text{-}1_{ij}, i \in [1..N], j \in [1..N]$$

$$Up_{ij} \leftrightarrow MatchZ_{ij} \bigwedge MatchY\text{-}1_{ij} \bigwedge MatchX_{ij}, i \in [1..N], j \in [1..N]$$

$$Down_{ij} \leftrightarrow MatchZ_{ij} \bigwedge MatchY{+}1_{ij} \bigwedge MatchX_{ij}, i \in [1..N], j \in [1..N]$$

$$Top_{ij} \leftrightarrow MatchZ\text{-}1_{ij} \bigwedge MatchY_{ij} \bigwedge MatchX_{ij}, i \in [1..N], j \in [1..N]$$

$$Bottom_{ij} \leftrightarrow MatchZ{+}1_{ij} \bigwedge MatchY_{ij} \bigwedge MatchX_{ij}, i \in [1..N], j \in [1..N]$$

Преобразования для равенства по всем координатам x:

$$MatchX_{ij} \leftrightarrow \bigvee_{a \in [1..L]} MatchX_{ija}, i \in [1..N], j \in [1..N]$$

$$MatchX{+}1_{ij} \leftrightarrow \bigvee_{a \in [1..L-1]} MatchX{+}1_{ija}, i \in [1..N], j \in [1..N]$$

$$MatchX\text{-}1_{ij} \leftrightarrow \bigvee_{a \in [2..L]} MatchX\text{-}1_{ija}, i \in [1..N], j \in [1..N]$$

По y:

$$MatchY_{ij} \leftrightarrow \bigvee_{a \in [1..L]} MatchY_{ija}, i \in [1..N], j \in [1..N]$$

$$MatchY{+}1_{ij} \leftrightarrow \bigvee_{a \in [1..L-1]} MatchY{+}1_{ija}, i \in [1..N], j \in [1..N]$$

$$MatchY\text{-}1_{ij} \leftrightarrow \bigvee_{a \in [2..L]} MatchY\text{-}1_{ija}, i \in [1..N], j \in [1..N]$$

По z:

$$MatchZ_{ij} \leftrightarrow \bigvee_{a \in [1..L]} MatchZ_{ija}, i \in [1..N], j \in [1..N]$$

$$MatchZ{+}1_{ij} \leftrightarrow \bigvee_{a \in [1..L-1]} MatchZ{+}1_{ija}, i \in [1..N], j \in [1..N]$$

$$MatchZ\text{-}1_{ij} \leftrightarrow \bigvee_{a \in [2..L]} MatchZ\text{-}1_{ija}, i \in [1..N], j \in [1..N]$$

Преобразования для равенства по фиксированной координате x:

$$Match\text{X}_{ija} \leftrightarrow X_{ia} \bigwedge X_{ja}, i \in [1..N], j \in [1..N], a \in [1..L]$$

$$Match\text{X+1}_{ija} \leftrightarrow X_{ia} \bigwedge X_{ja+1}, i \in [1..N], j \in [1..N], a \in [1..L-1]$$

$$Match\text{X-1}_{ija} \leftrightarrow X_{ia} \bigwedge X_{ja-1}, i \in [1..N], j \in [1..N], a \in [2..L]$$

По y:

$$Match\text{Y}_{ija} \leftrightarrow Y_{ia} \bigwedge Y_{ja}, i \in [1..N], j \in [1..N], a \in [1..L]$$

$$Match\text{Y+1}_{ija} \leftrightarrow Y_{ia} \bigwedge Y_{ja+1}, i \in [1..N], j \in [1..N], a \in [1..L-1]$$

$$Match\text{Y-1}_{ija} \leftrightarrow Y_{ia} \bigwedge Y_{ja-1}, i \in [1..N], j \in [1..N], a \in [2..L]$$

По z:

$$Match\text{Z}_{ija} \leftrightarrow Z_{ia} \bigwedge Z_{ja}, i \in [1..N], j \in [1..N], a \in [1..L]$$

$$Match\text{Z+1}_{ija} \leftrightarrow Z_{ia} \bigwedge Z_{ja+1}, i \in [1..N], j \in [1..N], a \in [1..L-1]$$

$$Match\text{Z-1}_{ija} \leftrightarrow Z_{ia} \bigwedge Z_{ja-1}, i \in [1..N], j \in [1..N], a \in [2..L]$$

Ограничение (2.2) выглядит как: $X_{00} \bigwedge Y_{00} \bigwedge Z_{00} \bigwedge X_{11} \bigwedge Y_{10} \bigwedge Z_{10}$, ограничение (2.3):

$$\bigwedge_{i \in [1..N]} \bigvee_{a \in [0..L]} (X_{ia})$$

$$\bigwedge_{i \in [1..N]} \bigwedge_{a \in [0..L]} \bigwedge_{b \neq a} (\neg X_{ia} \bigvee \neg X_{ib})$$

$$\bigwedge_{i \in [1..N]} \bigvee_{a \in [0..L]} (Y_{ia})$$

$$\bigwedge_{i \in [1..N]} \bigwedge_{a \in [0..L]} \bigwedge_{b \neq a} (\neg Y_{ia} \bigvee \neg Y_{ib})$$

$$\bigwedge_{i \in [1..N]} \bigvee_{a \in [0..L]} (Z_{ia})$$

$$\bigwedge_{i \in [1..N]} \bigwedge_{a \in [0..L]} \bigwedge_{b \neq a} (\neg Z_{ia} \bigvee \neg Z_{ib})$$

И, наконец, ограничение (2.4): $\bigwedge_{i \in [1..N]} Neighbor_{ii+1}$.

Осталось задать выражения для энергии, аналогично тому, как это было сделано для программирования в ограничениях.

Способ 1 Чтобы модуль энергии конфигурации был не меньше K, необходимо выполнение следующего условия. Если существует множество HH-пар мощности Z для данной последовательности, необходимо, чтобы существовало некоторое сочетание длины K из этого множества, все пары в котором являются соседями (C_Z^K — число сочетаний из Z по K): $\bigvee_{t=1..C_Z^K} \bigwedge_{k\in\{i_{t1}..i_{t_K}\}}(H_k \bigwedge Neighbor_k)$

Способ 2 По аналогии с переменными X_{ia}, Y_{ia}, Z_{ia} запишем требование к существованию $Begin_{ia}$, End_{ia}:
$\bigwedge_{i\in[1..K]} \bigvee_{a\in[1..N]}(Begin_{ia}a)$,
$\bigwedge_{i\in[1..K]} \bigvee_{a\in[1..N]}(End_{ia}a)$.
Условие единственности в данном случае проистекает из других ограничений и не требует введения дополнительных выражений.
Выражение (2.5) будет выглядеть следующим образом:
$\bigwedge_{i\in[1..K]} \bigvee_{a\in[1..N-3]} \bigwedge_{b<=a+2}(\neg Begin_{ia} \bigvee \neg End_{ia})$,
а выражение (2.6):
$\bigwedge_{i\in[1..K]} \bigwedge_{a\in[1..N]} \bigwedge_{b>a}(\neg Begin_{ia} \bigvee \neg Begin_{ib})$.
Чтобы записать выражение (2.7) в КНФ потребуется ввести дополнительные переменные для $i \in [1...K], j \in [1...K]$:

$MatchBegin_{ij}$ — верно ли, что $begin[i] = a, begin[j] = a \forall a \in [1...N]$,

$MatchBegin_{ija}$ — верно ли, что $begin[i] = a, begin[j] = a$,

$EndLess_{ij}$ — верно ли, что $end[i] < end[j] \forall a \in [1...N]$.

Через $Begin_{ia}$, End_{ia} для $i \in [1...K], j \in [1...K]$ эти переменные выражаются как:

$MatchBegin_{ij} \leftrightarrow \bigvee_{a\in[1..N]} MatchBegin_{ija}$,

$MatchBegin_{ija} \leftrightarrow Begin_{ia}a \bigwedge Begin_{ja}$,

$EndLess_{ij} \leftrightarrow \bigvee_{a\in[1..N]} End_{ia} \bigwedge_{b\le a} End_{jb}$.

Таким образом, выражение (2.7) записывается как

$$\bigwedge_{i\in[1..K]} \bigwedge_{j<i} \bigwedge_{a\in[0..N]} (\neg MatchBegin_{ij} \bigvee EndLess_{ij})$$

Все вышеописанные выражения, использующие операторы следствия и взаимно однозначного соответствия, преобразуются в КНФ по известным формулам:

$$A \leftrightarrow B \Leftrightarrow (\neg A \bigvee B) \bigwedge (A \bigvee \neg B)$$

$$A \rightarrow B \Leftrightarrow \neg A \bigvee B$$

В результате требуется $O(N^2 L)$ переменных для того, чтобы выразить ограничения по координатам, и $O(K^2 N_H)$ — чтобы выразить ограничения на энергию вторым способом, где N_H — число гидрофобных остатков в последовательности. В случае записи ограничений на энергию первым способом нет нужды в дополнительных переменных, но зависимость числа дизъюнктов в формуле от N_H становится экспоненциальной из-за наличия перебора по сочетаниям, в то время как второй способ порождает $O(K^2 N_H^2)$. Число дизъюнктов в формуле для записи ограничений на координаты, с учетом преобразования в КНФ, равно $O(N^2 L^2)$.

2.4 Поиск оптимального решения

Поскольку вышеизложенный алгоритм находит решение для фиксированного значения энергии, а в исходной задаче требуется найти максимальное по модулю, требуется подобрать такие параметры L и K, для которых полученная конфигурация будет оптимальной.

Предварительные исследования показали, что хорошим начальным значением для стороны решетки является $L = \sqrt{N} + 1$ для двумерного случая, и $L = \sqrt[3]{N} + 1$ для трехмерного. Это объясняется тем, что высокое число HH связей предполагает наличие гидрофобного ядра в свернутой цепочке. При достижении неудовлетворимой формулы L необходимо увеличить, чтобы убедиться, что на большем пространстве не существует решения с требуемыми параметрами.

Что же касается K, то его необходимо перебирать. После каждого запуска подсчитывается энергия полученной конфигурации и, если она больше текущей, ее инкрементированное значение следует использовать для следующего запуска. Этот подход позволяет быстро «проскочить» начальные значения.

Чем ближе значение энергии к оптимальному, тем больше времени уходит на поиск решения, поскольку число их возможных вариантов сокращается, и потому начальные запуски не занимают много времени. Тем не менее, стартовое ограничение энергии снизу с помощью какого-либо заведомо быстрого (пусть и не точного) метода (например, из области эволюционных алгоритмов), может помочь уменьшить число требуемых запусков и, тем самым, сократить суммарное время, требуемое на поиск оптимальной конфигурации.

2.5 Генерация всех решений

Язык *MiniZinc* предполагает, что программные инструменты, принимающие его на вход, обладают функциональностью для генерации всех возможных решений, удовлетворяющих данным ограничениям. Но в случае задачи удовлетворимости булевой формулы поиск всех решений — это уже другая, отдельная задача, называемая *ALL SAT*, и стандартные SAT-солверы эту возможность не поддерживают. Тем не менее, существует способ и с их помощью получать все решения. Он заключается в том, чтобы перезапускать солвер, включая в формулу отрицание предыдущего решения, пока она остается удовлетворимой.

Пусть на первом запуске были получены массивы координат $x = [x_1, ..., x_N]$ и $x = [y_1, ..., y_N]$. Тогда, чтобы исключить это решение, к формуле следует добавить выражение

$$\neg X_{ix_1} \bigvee \neg Y_{iy_1} \bigvee ... \bigvee \neg X_{ix_N} \bigvee \neg Y_{iy_N}$$

Таким образом, решения исключаются по принципу фиксации координат. Однако в этом случае требуется либо обязательно фиксировать начало последовательности 2.2 (что может быть не всегда желательно), либо исключать также все возможные сдвиги и повороты для набора координат. Кроме того, этот способ генерирует также решения, отличающиеся только положением полярных остатков, не связанных ограничениями на энергию, и их число может быть значительным. В случае, если интересно рассмотреть именно решения, отличающиеся именно конфигурацией гидрофобного ядра, для второго способа генерации энергии можно фиксировать переменные $begin = [b_1, ..., b_k]$

и $end = [e_1, ..., e_K]$:

$$\neg Begin_{ib_1} \bigvee \neg End_{ie_1} \bigvee ... \bigvee \neg Begin_{ib_K} \bigvee \neg End_{ie_K}$$

Такой подход также автоматически исключает все геометрически эквивалентные решения.

2.6 Эвристики

Некоторыми особенностями задачи можно воспользоваться для того, чтобы сократить число переменных в булевой формуле, либо наложить дополнительные ограничения, чтобы сократить время поиска решения. Например, можно заметить, что в соседних узлах решетки могут быть расположены только такие остатки i и j, разница индексов которых в последовательности нечетна. Поскольку $Neighbor_{ij}$ никогда не может быть верным для пар индексов с четной разностью, для них нет смысла вводить соответствующие переменные (а так же все, которые связаны с $Neighbor_{ij}$ или выражаются через них). Кроме того, даже из этого множества пар индексов свойство $Neighbor_{ij}$ требуется не для всех: нас интересуют только последовательные пары (i и $i+1$), а также те, в которых оба остатка являются гидрофобными (i и j где $H_{ij} = true$).

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

В данной главе приводятся подробности реализации алгоритмов, описанных в главе 2, результаты экспериментальных исследований производительности и точности их работы, а также их сравнение с некоторыми из существующих методов.

3.1 Реализация

В процессе работы были использованы языки программирования *Java* и Groovy, инструменты *Lingeling* и *Glucose* для решения задачи о выполнимости булевых формул, *Opturion* для решения CSP.

Lingeling и *Glucose* принимают на вход булевы формулы в коньюнктивной нормальной форме, записанные в формате *DIMACS*. На *Java* было реализовано формирование этих данных по исходной последовательности и параметрам, задающим размер решетки и нижнюю границу энергии.

Перебор параметров для всех методов был реализован на Groovy, как и формирование входных данных на языке *MiniZinc*, который принимается на вход инструментом *Opturion*.

3.1.1 Выбор инструмента

Изначально *Lingeling* и *Glucose* были выбраны для решения данной задачи потому, что они занимали призовые места в последних соревнованиях, проводимых для определения лучших SAT-солверов [29]. Для них были произведены предварительные тесты, результаты которых приведены на рис.3.1. Для двадцати различных тестовых последовательностей показано время работы обоих инструментов по логарифмической шкале. Можно заметить, что для данной задачи *Lingeling* превосходит своего конкурента в каждом из слу-

чаев. Поэтому во всех описанных далее экспериментах был использован именно этот инструмент.

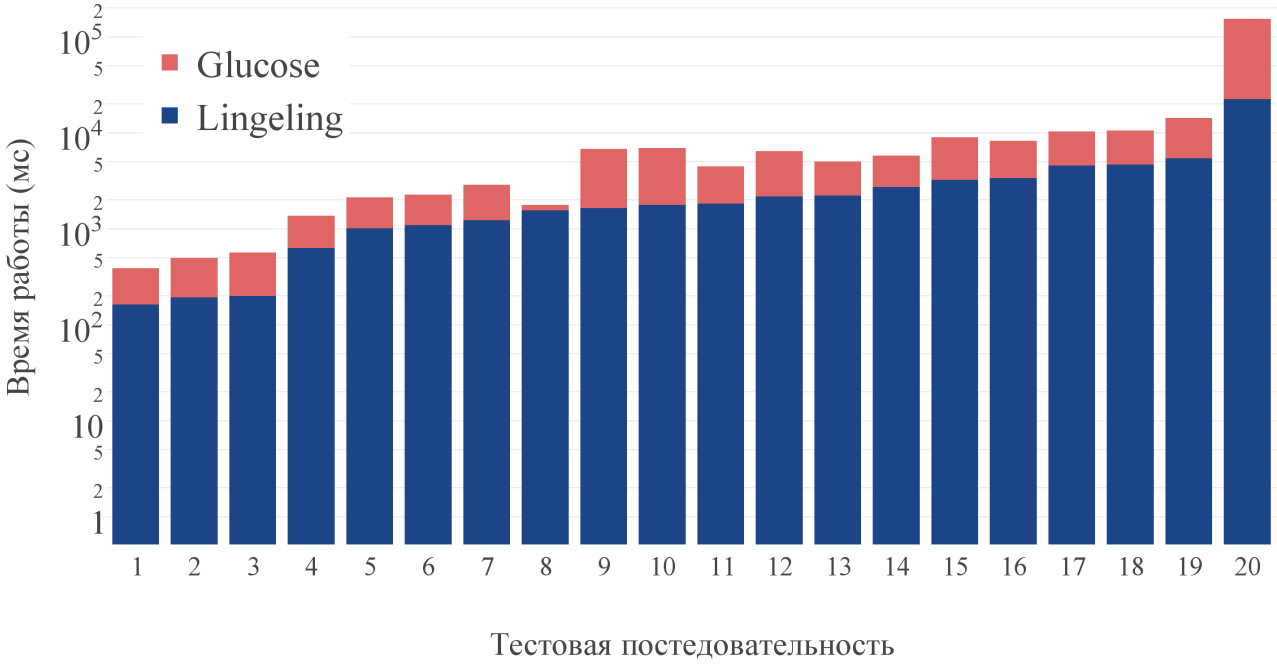


Рис. 3.1: Время работы SAT-солверов

3.1.2 Тестовое окружение

Запуск тестов производился на компьютере с процессором Intel(R) Core(TM) i5 CPU 760 @ 2.80GHz и 4Гб ОЗУ, под операционной системой Ubuntu 12.04.2.

3.2 Сравнение производительности реализованных алгоритмов

Алгоритмы, описанные в главе 2, фигурируют в данном разделе и далее под именами:

- *CSP1* — сведение к задаче удовлетворения ограничений, первый способ задания энергии (аналогичен решению, предлагаемому в [18]);
- *CSP2* — сведение к задаче удовлетворения ограничений, второй способ задания энергии;

- *SAT1* — сведение к задаче удовлетворимости булевой формулы, первый способ задания энергии;
- *SAT2* — сведение к задаче удовлетворимости булевой формулы, второй способ задания энергии.

В табл. 3.1 представлены результаты запусков всех вышеописанных алгоритмов на случайно сгенерированных данных. Было сгенерировано 10 последовательностей каждой длины, с соотношением гидрофобных и полярных остатков один к одному. Время работы в миллисекундах усреднено по всем запускам для этих последовательностей. Знак > перед числом означает, что для некоторых запусков было превышено ограничение по времени работы, и при усреднении для них использовалось пороговое значение 600000 мс.

Таблица 3.1: Среднее время работы для случайных последовательностей

Длина	**SAT2**	**SAT1**	**CSP2**	**CSP1**
17	178	641	1457	1357
18	458	1643	2498	2634
19	653	2356	6896	7290
20	1303	4646	6741	8013
21	6427	35241	43570	47782
22	21346	128084	83981	100784
23	32673	190967	135794	158738
24	49049	>420781	>457218	>560344
25	52369	>430013	>510236	>598523
26	75698	>552816	>573698	>598524

В табл. 3.2 приведено среднее время работы на случайно сгенерированных последовательностях длины 22 в зависимости от числа гидрофобных остатков. Как и в предыдущем эксперименте, значения в каждой строке — среднее арифметическое для десяти запусков на различных входных данных с заданными характеристиками.

Таблица 3.2: Среднее время работы для разного числа H

Число H	**SAT2**	**SAT1**	**CSP2**	**CSP1**
4	139	143	494	508
6	237	268	495	560
8	377	944	1235	1546
11	3569	17173	45631	54892
13	4766	36646	72076	92365
15	5682	74982	112364	123566
17	7895	292075	230455	256899
19	10653	605239	365789	456881

Из результатов экспериментов можно сделать вывод, что алгоритм *SAT2* превосходит остальные алгоритмы по производительности, а время его работы растет с увеличением размерности задачи значительно медленнее. Кроме того, алгоритм *CSP2* показал себя более эффективным, чем алгоритм *CSP1*. Что же касается алгоритма *SAT1*, то, хотя он показал неплохую на тестовых примерах меньшей размерности, то с увеличением числа гидрофобных остатков его производительность резко ухудшилась. Этот результат совпадает с ожидаемым при теоретическом анализе: число дизъюнктов в генерируемой формуле экспоненциально зависит от числа гидрофобных остатков.

3.3 Сравнение с другими алгоритмами

В данном разделе приводятся данные сравнения производительности и точности работы реализованных алгоритмов по сравнению с некоторыми из уже существующих методов.

3.3.1 Сравнение с перебором с вероятностными отсечениями

Использовать абсолютные значения времени работы алгоритмов, указанные в статьях, представляется затруднительным, поскольку программы реализуются на разных языках, а запуски выполняются на совершенно различных машинах. Алгоритм перебора с вероятностными отсечениями, кратко описанный в главе 1, отличается простотой реализации, и потому был выбран

для сравнения в производительности с предлагаемыми в данной работе алгоритмами.

Вероятность отсечения ветвей дерева решений в этом алгоритме регулируется двумя параметрами, которые для данной серии испытаний были выбраны согласно рекомендации авторов: $p_1 = 0.8$, $p_2 = 0.5$. Тестирование осуществлялось на автоматически сгенерированных тестовых последовательностях, а сравнение производилось с лучшим из предложенных алгоритмов ($SAT2$). На рис. 3.2 можно видеть среднее время работы каждого из алгоритмов по десяти запускам для каждой длины последовательности.

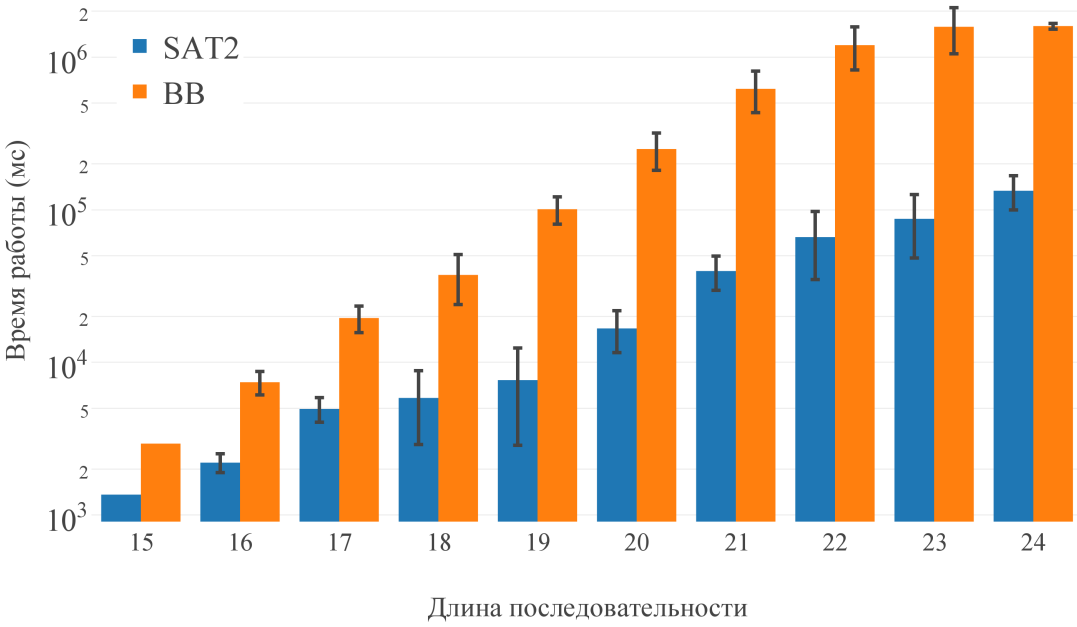


Рис. 3.2: Сравнение B&B и SAT2

Несмотря на вероятностную природу алгоритма, с которым осуществлялось сравнение, для такой, сравнительно небольшой, длины последовательности он показал хорошую точность, однако время его работы оказалось в большинстве случаев много больше, чем для алгоритма $SAT2$.

Тестирование для других соотношений гидрофобных и полярных остатков показало, что исключение могут составить лишь последовательности, в которых относительное число гидрофобных остатков очень велико (около 0.9), и при параметрах алгоритма, ведущих к отсечению большего числа ветвей. Причина этого в том, что алгоритм перебора отсекает ветви дерева решений именно при размещении этих остатков, а число переменных и выражений при сведении к SAT, напротив, растет пропорционально

числу возможных HH-пар. Однако даже в этом случае есть риск потерять оптимальное решение для более длинных последовательностей, чего не может случиться с предлагаемым в данной работе алгоритмом.

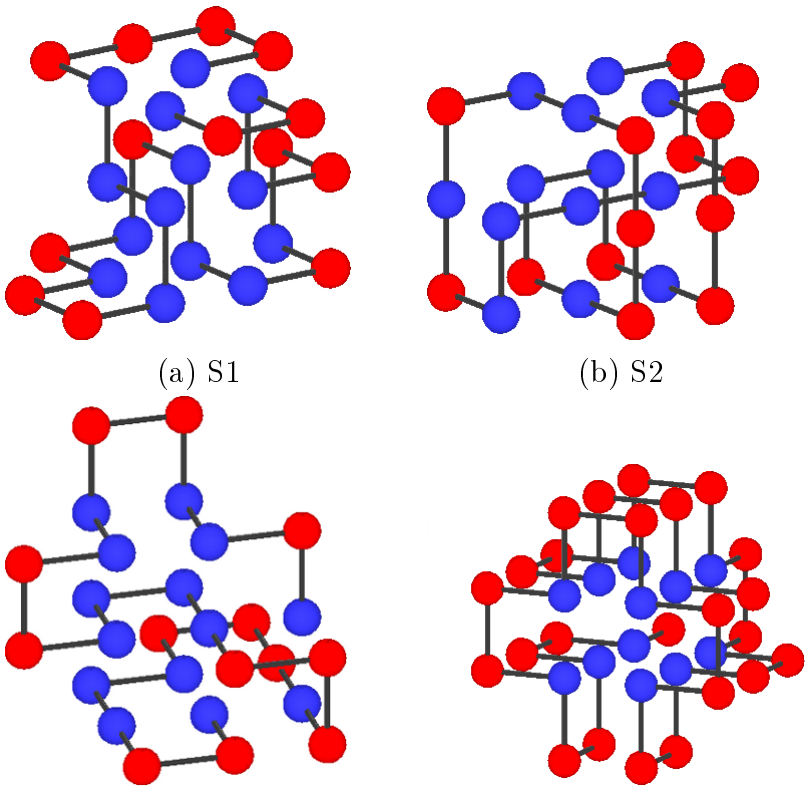
3.3.2 Сравнение с альтернативной реализацией использования программирования в ограничениях

Вариант использования программирования в ограничениях, предложенный в статье [18], фигурирует в данной работе под именем *CSP1* и отличается от *CSP2* способом задания ограничений на энергию. Несмотря на то, что авторы ставят своей целью решение несколько иной задачи (их модель содержит более чем два вида аминокислотных остатков, и функция энергии имеет более сложный вид), они также приводят результаты использования своего решения для нескольких тестовых HP-последовательностей, приведенных в табл. 3.3.

Таблица 3.3: Тестовые последовательности

#	**Последовательность**	**Длина**	E_{min}
S1	HPPPPHHHHPPHPHPHHHPHPPHHPPH	27	16
S2	HPPPHHHHPHPHHPPPHPHHPHPPPHP	27	15
S3	HPHHPPHHPPHHHHPPPHPPPHHHPPH	27	16
S4	HHPHHPHHPHHHHHHPPHHHHHHPPHHHHHHH	31	20
S5	PHPPHPPHPPHPPHPPHPPHPPHPPHPPHPPHPPHPPHP	36	28

Этот набор предполагает решение трехмерного варианта задачи. Конфигурации с оптимальной энергией, полученные для первых четырех последовательностей, приведены на рис. 3.3. Прямое сравнение производительности с реализацией, описанной в статье, не представляется возможным, однако рассматриваемая в данной работе реализация *CSP1* превзошла таймаут в 20 минут даже для первых трех, более коротких последовательностей, в то время как *SAT2* отработал за 3-4 минуты. Отсюда можно сделать вывод, что предлагаемый алгоритм *SAT2* является более эффективным для решения этой задачи.



(a) S1 (b) S2

(c) S3 (d) S4

Рис. 3.3: Оптимальные конфигурации

3.3.3 Сравнение с различными эволюционными алгоритмами для двумерного случая

Для тестирования решений для двумерного варианта задачи в литературе используются последовательности из табл. 3.4. Поскольку результаты, получаемые с помощью эволюционных алгоритмов, не являются гарантированно оптимальными, их сравнение производят по наилучшему значению энергии, полученному для данной последовательности.

Таблица 3.4: Стандартные тестовые последовательности

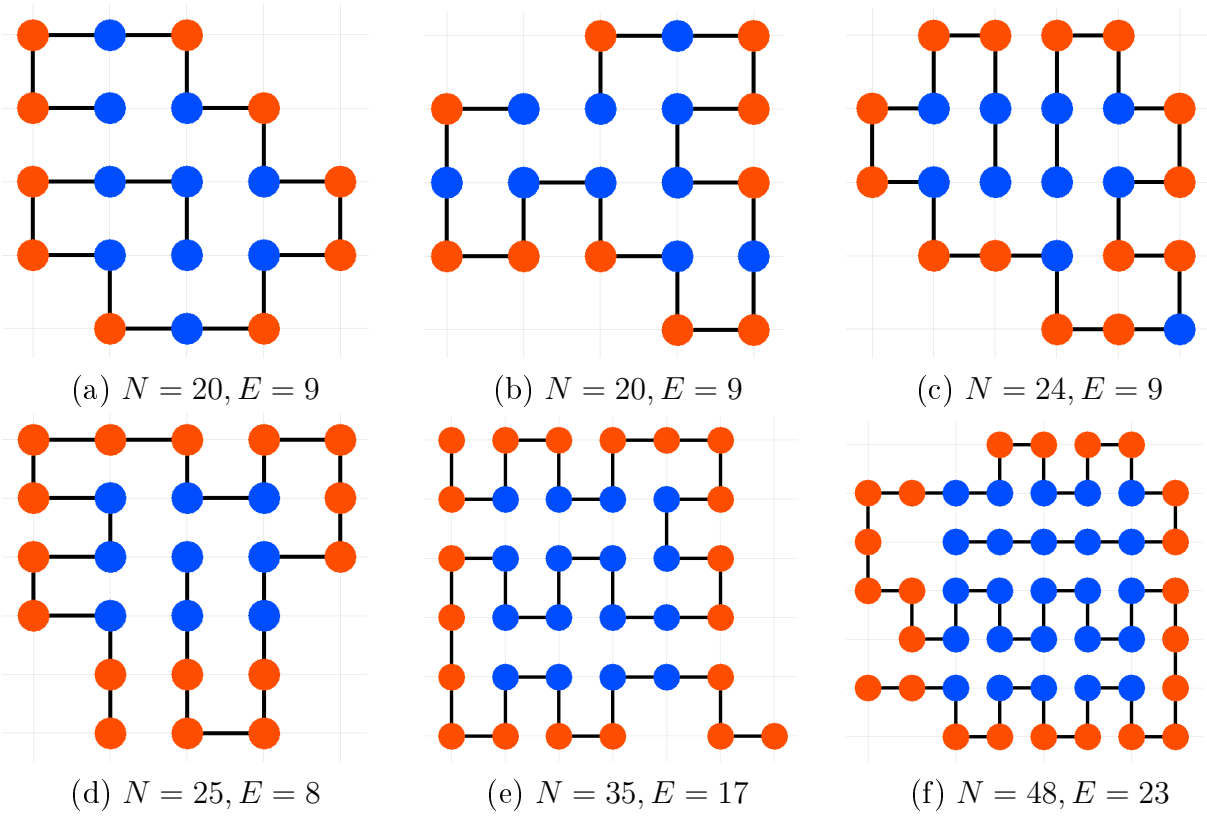
#	N	Последовательность
1	20	HHHPPHPHPHPPHPHPHPPH
2	20	HPHPPHHPHPPHPHHPPHPH
3	24	HHPPHPPHPPHPPHPPHPPHPPHH
4	25	PHPPHHPPPPHHPPPPHHPPPPHH
5	36	PPHHPPHHPPPPPHHHHHHHHPPHHPPPPHHPPHPP
6	46	PHHHPHHHPPPHPHHPHHPPHPHHHHPHPPHHHHHPHPHHPP-HHP
7	48	PHPPHHPPHHPPPPPHHHHHHHHHHPPPPPPHHPPHHPPHPP-HHHHH
8	50	HHPHPHPHPHHHHPHPPPHPPPHPPPPHPPPHPPPHPHHHHP-HPHPHPHH

Поскольку предлагаемый метод по определению всегда находит глобально оптимальное значение, такое сравнение не вполне оправдано. Тем не менее, в табл. 3.5 можно видеть результаты для таких методов, как Монте-Карло, генетические алгоритмы, смешанные стратегии поиска и поиск с вероятностными отсечениями. Нетрудно заметить, что для каждого из этих методов существует последовательность, для которой не был найден глобально оптимальный результат.

Таблица 3.5: Лучшая полученная энергия для различных алгоритмов(двумерный случай)

SAT2	**MC** [21]	**GA** [21]	**MS** [35]	**BB** [20]
9	9	9	9	9
9	9	9	9	9
8	8	7	8	8
14	12	14	14	14
23	18	22	22	22
21	19	21	21	21

Схемы оптимальных конфигураций, полученные методом сведения к задаче о выполнимости булевых формул для первых шести последовательностей из списка, приведены на Рис. 3.4.



(a) $N = 20, E = 9$ (b) $N = 20, E = 9$ (c) $N = 24, E = 9$

(d) $N = 25, E = 8$ (e) $N = 35, E = 17$ (f) $N = 48, E = 23$

Рис. 3.4: Оптимальные конфигурации

3.3.4 Сравнение с различными эволюционными алгоритмами для трехмерного случая

Ряд стандартных тестовых последовательностей длины 27, используемых для тестирования алгоритмов, решающих трехмерный вариант задачи, приведен в табл. 3.6.

Таблица 3.6: Тестовые последовательности

#	**Последовательность**	**Длина**
273d.1	PHPHPHHHPPHPHPPPPPPPPPPHHP	27
273d.2	PHHPPPPPPPPPPHHPPHHPPHPPHPH	28
273d.3	HHHHPPPPPHPPPPPHHHPPPPPPPPH	29
273d.4	HHHPPHHHHPPPHPHPPHHPPHPPPHH	30
273d.5	HHHHPPPPHPHHPPPHHPPPPPPPPPP	31
273d.6	HPPPPPPHPHHHPPHHPPPHPPPPHPH	32
273d.7	HPPHPHHPPPHPPPPPHPHHPHPHPHH	33
273d.8	HPPPPPPPPPPPHPHPPPPPPPPHPHH	34
273d.9	PPPPPPPHHHPPPHPHHPPPHPPHPPP	35
273d.10	PPPPPHHPHPHPHPHPPHHPHHPHPPP	36

В табл. 3.7 можно видеть лучшие значения энергии, полученные для этих последовательностей различными реализациями генетических алгоритмов, а также методом дифференциальной эволюции. Для этих примеров время, затраченное алгоритмом $SAT2$ для поиска глобально оптимальной конфигурации, составило от 0.5 до 46 секунд (максимум приходится на последовательность 273d.7, для которой максимально возможное число HH-соединений велико).

Таблица 3.7: Лучшая полученная энергия для различных алгоритмов (трехмерный случай)

SAT2	**J&K GA** [13]	**P&P GA** [36]	**U&M GA** [21]	**DE** [9]
9	9	9	9	9
10	10	10	9	10
8	8	8	8	8
15	15	15	15	15
8	8	8	8	8
12	11	11	11	11
13	13	13	12	13
4	4	4	4	4
7	7	7	7	7
11	11	11	11	11

В табл. 3.8 приведены оптимальные конфигурации для этих последовательностей, закодированные поворотами в пространстве:

L — влево, R — вправо,

U — вперед, D — назад,

T — вверх, B — вниз.

Таблица 3.8: Оптимальные конфигурации для трехмерного случая

#	**Конфигурация**
273d.1	LTUUBLTTDBDBLLURUBRBDRTTLB
273d.2	UBLBBRURRTTTLDRBLUBLTUTDLB
273d.3	BDTDRBUTUBBDLLTUTUBLDDTDRU
273d.4	DRRULULLBRRDDLBUTLBDTLUTRD
273d.5	BLTTLDBRTRBDLBURRTRTTLTDBB
273d.6	LUUTDDRUBBDDRUTRBULULTRDTD
273d.7	BLTDTTDRBUBBLLTTURRTDRBBUT
273d.8	TTTRULUBDRDBURRUBULDDLDRTT
273d.9	DBUULDTDLDLUUTRBURTDRDLLTD
273d.10	TDDBLLDRBURUTLUTDDLUBBRURB

3.4 Поиск всех возможных оптимальных конфигураций

Инструменты, решающие задачу удовлетворения ограничений, поддерживают возможность генерации всех решений, удовлетворяющих ограничениям. Кроме того, был реализован описанный в разделе 2.5 способ получения всех решений для сведения к задаче выполнимости булевых формул. Реализация была протестирована на ряде примеров из числа стандартных тестовых последовательностей, приведенных выше, а также на случайно сгенерированных данных.

На Рис. 3.5 приведены шесть из девяти различных оптимальных решений, сгенерированных для последовательности 3 из табл. 3.4.

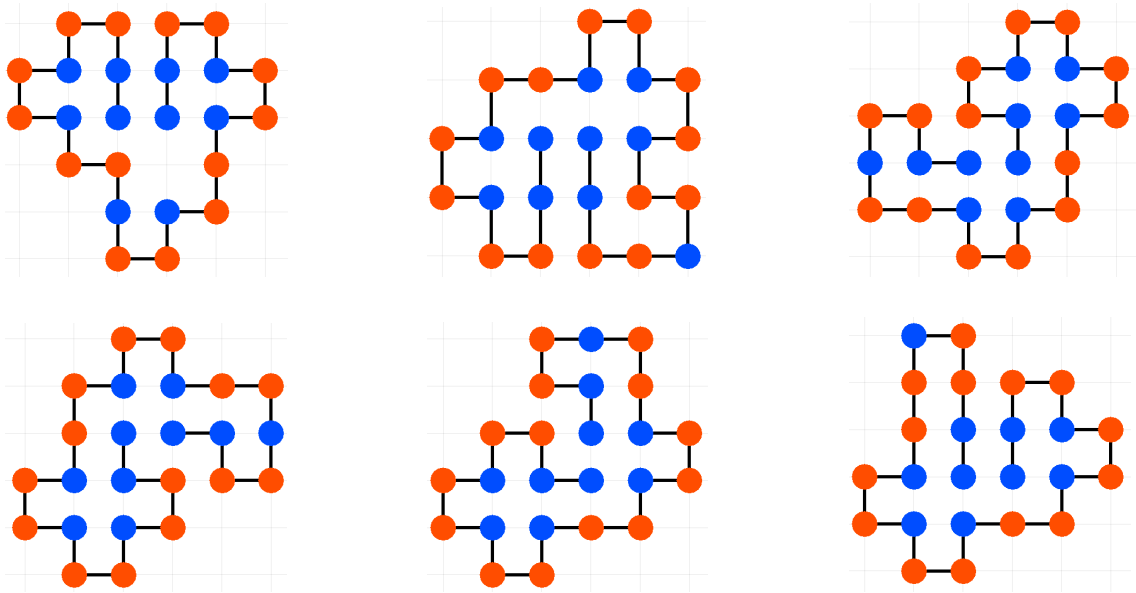


Рис. 3.5: Шесть различных решений для последовательности 3 из табл. 3.4

Результаты сравнения времени работы алгоритмов *SAT2* и *CSP2* при генерации всех решений для случайных последовательностей длины 22, усредненные по десяти запускам для каждого числа решений, можно видеть на рис. 3.6. Нетрудно заметить, что такой способ использования SAT является неоптимальным, поскольку программное средство каждый ищет решение заново, пусть и с дополнительными ограничениями. Кроме того, большая часть времени уходит на последний запуск, когда формула становится неудовлетворимой. Отсюда можно сделать вывод, что для данной модификации задачи программирование в ограничениях является предпочтительным методом.

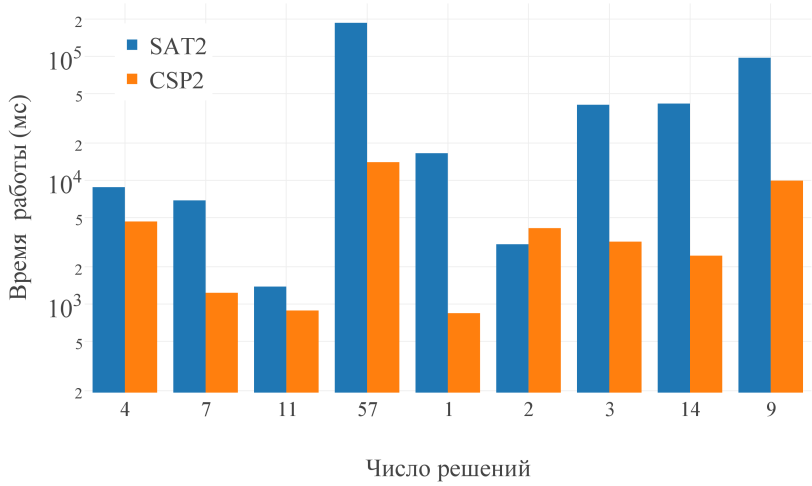


Рис. 3.6: Сравнительное время работы *SAT2* и *CSP2* для генерации всех решений

ЗАКЛЮЧЕНИЕ

Данная диссертация направлена на разработку алгоритмов решения задачи поиска оптимальной конфигурации гидрофобно-полярной модели протеина с использованием задачи удовлетворения ограничений и задачи выполнимости булевой формулы. Предлагаемый метод отличается тем, что находит глобально оптимальное решение, для поиска которого используются сторонние программные средства. Постоянное совершенствование этих инструментов позволяет увеличивать эффективность полученного программного продукта без изменения его исходного кода, путем замены используемого солвера на новую, более производительную версию.

В ходе работы был проведен аналитический обзор методов, используемых для решения данной задачи. Обзор показал, что методы решения задачи удовлетворимости булевой формулы для решения этой задачи ранее не применялись, а существующий способ использования программирования в ограничениях отличается о того, что предлагается в данной работе.

В результате был разработан и реализован алгоритм сведения исходной задачи к виду, принимаемому на вход сторонними программными средствами. Также было показано преимущество в производительности предлагаемых алгоритмов по сравнению с двумя другими методами: существующим способом использования программирования в ограничениях, а также с одним из вариантов перебора с отсечениями, и показано их превосходство в точности работы по сравнению с некоторыми эволюционными алгоритмами. Кроме того, были исследованы возможности использования предлагаемых алгоритмов для генерации всех возможных решений.

По теме диссертации был сделан доклад на III Всероссийском конгрессе молодых ученых (8-12 апреля 2014 года, НИУ ИТМО).

ЛИТЕРАТУРА

1. B. Alberts D. B. e. a. Essential Cell Biology. New York: Birkhauser Boston, 2010. P. 120–170.

2. Xiang Z. Advances in homology protein structure modeling. 2006. Vol. 7, no. 3. P. 217–227.

3. P. Privalov G. M. Contribution of hydration to protein folding thermodynamics. 1993. Vol. 232, no. 11. P. 660–679.

4. Rosetta System. URL: https://www.rosettacommons.org/.

5. J. T. Ngo J. Marks M. K. The Protein Folding Problem and Tertiary Structure Prediction. Birkhauser Boston, 1994. P. 433–506.

6. J. Santos P. Villot M. D. Protein Folding with Cellular Automata in the 3D HP Model // Proc. of 15th annual conference companion on Genetic and evolutionary computation. 2013. P. 1595–1602.

7. Fidanova S. 3D HP Protein Folding Problem using Ant Algorithm // Proc. of BioPS International Conference, Sofia, Bulgaria, III. 2006. P. 19–26.

8. Dorigo M., Gambardella L. M. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem // Trans. Evol. Comp. Piscataway, NJ, USA, 1997. Vol. 1, no. 1. P. 53–66.

9. H. S. Lopes R. B. A Differential Evolution Approach for Protein Folding Using a Lattice Model. 2007. no. 6. P. 905–908.

10. H. Lu G. Y. Extremal Optimization for protein folding simulations on the lattice. 2009. Vol. 57. P. 1855–1861.

11. H. P. Hsu V. Mehra W. N. P. G. Growth Alogorithm for Lattice Heteropolymers at Low Temperature // Chemical Phisics. 2003. P. 444–451.

12. N. Mansour F. Kanj H. K. Search Algorithms and Applications. InTech, 2011. P. 69–79.

13. C. M. Johnson A. K. A genetic algorithm with backtracking for protein structure prediction // Proceedings of the 8th International Conference on Genetic Algorithms. 2006. P. 299–300.

14. L. Toma S. T. Contact interactions method: A new algorithm for protein folding simulations. 1996. no. 5. P. 147–153.

15. K. Yue K. D. Forces of Tertiary Structural Organization in Globular Proteins // Nat. Acad. Sci., USA. 1995. P. 146–150.

16. K. Dill K. M. Fiebig H. S. C. Cooperativity in Protein-folding Kinetics // Nat. Acad. Sci., USA. 1993. P. 1942–1946.

17. T. Beutler K. D. A Fast Conformational Method: A New Algorithm for Protein Folding Simulations // Protein Science. 1996. P. 147–153.

18. R. Backofen S. Will E. B.-B. Application of Constraint Programming Techiniques for Structure Prediction of Lattice Proteins with Extended Alphabets. 1999. Vol. 15, no. 3. P. 234–242.

19. R. Babbush A. Perdomo-Ortiz B. O. W. M. A. A.-G. Construction of Energy Functions for Lattice Heteropolymer Models: A Case Study in Constraint Satisfaction Programming and Adiabatic Quantum Optimization. 2013. P. 234–242.

20. M. Chen W. Q. H. A branch and bound algorithm for the protein folding problem in the HP lattice model. 2005. no. 3. P. 225–230.

21. R. Unger J. M. Contact interactions method: A new algorithm for protein folding simulations. 1993. no. 23. P. 75–81.

22. F. Liang W. H. W. Evolutionary Monte Carlo for protein folding simulations. 2001. Vol. 115, no. 7. P. 3374–3380.

23. Heule M., Verwer S. Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications / Ed. by J. Sempere,

P. García. Springer Berlin Heidelberg, 2010. Vol. 6339 of *Lecture Notes in Computer Science*. P. 66–79.

24. Панченко Е.В. Ульянцев В.И. Применение методов решения задачи о выполнимости квантифицированной булевой функции для построения управляющих конечных автоматов по сценариям работы и темпоральным свойствам. 2013. Т. 86, № 4. С. 151–153.

25. В.И. Ульянцев. Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы. 2012. Т. 77, № 1. С. 96–100.

26. В.И. Ульянцев. Метод построения управляющих конечных автоматов по сценариям работы на основе решения задачи удовлетворения ограничений. СПб, 2013. С. 256–260.

27. Хопкрофт Джон Э., Мотвани Раджив, Ульман Джеффри Д. Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.

28. Olga Ohrimenko Peter J. Stuckey M. C. Principles and Practice of Constraint Programming – CP 2007 // 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings. 2007. P. 544–558.

29. Sat Competition. URL: http://www.satcompetition.org/.

30. Lingeling SAT Solver. URL: http://fmv.jku.at/lingeling/.

31. Glucose SAT Solver. URL: http://www.labri.fr/perso/lsimon/glucose/.

32. MiniZinc Challenge. URL: http://www.minizinc.org/challenge.html.

33. Opturion CPX solver. URL: http://opturion.com/.

34. Walsh T. SAT v CSP // Proc. of the 6th International Conference on Principles and Practice of Constraint Programming (CP-00). Springer-Verlag, 2000. P. 441–456.

35. H. Jing F. Jing S. Z. Mixed search algorithm for protein folding // Wuhan University Journal of Natural Sciences. 2003. Vol. 8, no. 3. P. 765–768.

36. A. L. Patton W.F. Punch E. D. G. A Standard GA Approach to Native Protein Conformation Prediction // Proceedings of the Sixth International Conference on Genetic Algorithms. Morgan Kaufmann, 1995. P. 574–581.