Методы повышения эффективности алгоритма полного перебора на примере решения задачи о неограниченном ранце

Ай Мин Тайк, С.А. Лупин, Мин Тху Кхаинг

***Аннотация*— Универсальным методом точного решения задач дискретной оптимизации, инвариантным к условиям поиска, является алгоритм полного перебора вариантов (BFA). Его широкое использование на практике, ограничивает высокая вычислительная сложность, определяемая размерностью пространства поиска. Это определяет интерес исследователей к поиску подходов к ограничению числа рассматриваемых вариантов решения без потери его точности. В статье рассматривается вопрос повышения эффективности алгоритма полного перебора вариантов при решении задач дискретной оптимизации. На примере задачи о неограниченном ранце (UKP) показана возможность реализации генератора, обеспечивающего исключение вариантов, не удовлетворяющих условиям задачи, без расчета критериальной функции. Проведена оценка реализуемости генераторов и для многопоточных BFA-приложений. Подтверждена эффективность предложенного подхода и для последовательных и для параллельных реализаций. Результаты вычислительных экспериментов совпадают с теоретическими расчетами. Четырех поточное приложение с оптимизированным генератором обеспечивает более чем 30-ти кратное ускорение вычислений. Исследуемый подход может применяться и для других задач дискретной оптимизации.**

***Ключевые слова*— алгоритм полного перебора, параллельная реализация алгоритмов оптимизации, задача о неограниченном ранце, OpenMP.**

I. Введение

Совершенствование алгоритмов оптимизации является сегодня актуальной темой исследований, поскольку они находят широкое применение для решения практических задач в различных автоматизированных системах. К одним из универсальных методов решения задач дискретной оптимизации, инвариантных к условиям поиска, можно отнести алгоритм полного перебора вариантов (BFA, Brute Force Algorithm, метод «грубой силы»). Он является фундаментальной методологией решения проблем в живой природе - методом проб и ошибок, который лежит в основе эволюции. Этот метод основан на исчерпывающем поиске оптимального решения в пространстве, содержащем все возможные варианты.

Статья получена 8 февраля 2023.

С.А. Лупин, профессор, Национальный исследовательский университет «МИЭТ», МСЦ РАН – филиал ФИЦ ФНЦ НИИСИ РАН, (e-mail: lupin@miee.ru);

Ай Мин Тайк, к.т.н., докторант Национального исследовательского университета «МИЭТ» (e-mail: ayeminthike52@gmail.com);

Мин Тху Кхаинг, аспирант Национального исследовательского университета «МИЭТ»; (email: minthukhaing55@gmail.com).

Новейшие компьютерные технологии, такие как многоядерные процессоры и ускорители, позволяют расширить область применения BFA [1].

Такой подход гарантирует нахождение точного решения задачи, не требует глубокого знания особенностей предметной области. На этом основана популярность BFA и его широкое использование на практике, которое ограничивает только высокая вычислительная сложность, определяемая размерностью пространства поиска.

Рассмотрим несколько примеров практического использования BFA.

Алгоритм полного перебора является основным инструментом для получения доступа к конфиденциальной информации. Наиболее распространенной атакой является кража или подбор паролей пользователей. Злоумышленник может использовать некоторый ограниченный алфавит для синтеза всех возможных вариантов паролей для атаки на защищаемый объект.

Для снижения трудоемкости процедуры поиска пароля хакеры используют словарь, содержащий наиболее часто встречающиеся комбинации символов. Известными инструментами, реализующими атаки на системы безопасности с использованием алгоритма полного перебора, являются: Aircrack-ng, John the Ripper, Rainbow Crack, L0phtCrack, Ophcrack, Hashcat, DaveGrohl, Ncrack, THC Hydra [2].

Другим примером использования BFA-стратегии можно считать алгоритм нахождения ближайшей пары в множестве из N вершин на двумерной декартовой плоскости [3]. Практическим примером такой задачи может служить система управления воздушным движением, где нужно непрерывно следить за расстояниями между летящими рядом самолетами. Правила безопасности полетов требуют, чтобы это расстояние было не меньше установленного минимума. В этом случае BFA вычисляет расстояние между всеми отслеживаемыми объектами и возвращает индексы тех самолетов, расстояние между которыми является наименьшим.

Еще одним примером, в котором BFA используется для решения задачи дискретной оптимизации, является алгоритм Гамильтона построения кратчайшего цикла в направленном графе (задача коммивояжера). В общем случае сложность нахождения решения для N вершин составляет $O(N!)$ [4].

К задачам дискретной оптимизации относится и задача о ранце (или рюкзаке), которая находит различные применения в области сетевой безопасности,

криптографии и искусственного интеллекта [5]. Существует множество вариантов ее постановки [6]. В частности, в работе [7] авторы представили решение задачи о бинарном рюкзаке с помощью BFA и динамического программирования. Отметим, что задача о ранце одна из немногих задач дискретной оптимизации, для которых решение может быть получено с помощью метода динамического программирования.

В статье мы рассмотрим некоторые методы повышения эффективности BFA при решении оптимизационных задач. В качестве примера будет использована популярная у исследователей задача о неограниченном ранце (Unbounded Knapsack Problem, UKP).

II. Алгоритм полного перебора и его варианты

Для задач дискретной оптимизации, для которых нет известного эффективного алгоритма, метод полного перебора вариантов остается единственным возможным инструментом нахождения точного решения. В задаче о ранце под вариантом решения понимается набор предметов, удовлетворяющий ограничению. Это дает возможность использования BFA для нахождения точного решения. Выше отмечена важная особенность BFA, ограничивающая его практическое использование – высокая трудоемкость, пропорциональная числу рассматриваемых вариантов решения. Это справедливо и для UKP.

Для снижения трудоемкости полного перебора вариантов традиционно используют **метод ветвей и границ** (Branch and Bound Method, BBM). Метод ветвей и границ – это парадигма проектирования алгоритмов, которая обычно используется для решения задач дискретной оптимизации [8]. В худшем случае при использовании BBM будут рассмотрены все возможные варианты, как и в BFA, а в лучшем случае решение будет найдено при спуске только по одной ветви дерева. Также как и BFA, BBM гарантирует нахождение точного решения, однако он значительно сложнее в реализации, особенно в параллельной версии, имеет высокие требования к памяти. Кроме того, эффективность метода определяется конструкцией оценки, ее способностью отсекать неперспективные ветви.

Другим подходом, обеспечивающим снижение трудоемкости полного перебора, являются методы, реализующие стратегию **случайного поиска** (Random Search Algorithm, RSA). Случайный поиск — это подход к решению задач численной оптимизации, в основе которого лежит генерация случайных вариантов решения и выбор из них наилучшего.

Как и BFA, RSA не требователен к памяти, просто реализуется в последовательном и параллельном вариантах, однако он дает возможность получить только удовлетворительное, а не точное решение.

Поскольку в RSA нет явного критерия завершения итераций, на практике используют два подхода. В первом случае задают время, по истечении которого вычисления завершаются, а во втором критерием останова является генерация заданного числа вариантов решения. В ряде случаев RSA может быть остановлен при достижении заданной точности решения [9].

Ещё одним фактором, снижающим эффективность RSA, является неконтролируемая повторяемость генерируемых вариантов решения. С этим обстоятельством пытаются бороться, но затраты на контроль повторяемости значительно превосходят получаемый эффект.

В этой статье мы исследуем возможность повышения эффективности только BFA, не сравнивая результаты с другими алгоритмами. Кроме того, поскольку трудоемкость вычисления значения критериальной функции для каждого сгенерированного варианта решения не зависит от метода его получения, наши усилия будут сосредоточены только на процессе самой генерации.

Из самых общих соображений можно предположить, что повысить производительность BFA можно с помощью следующих методов:

1 - повысить скорость генерации вариантов;

2 - использовать генератор, учитывающий особенности задачи;

3 - исключать неперспективные варианты без расчета критериальной функции.

Хотя эти методы инвариантны к решаемой оптимизационной задаче, в каждом случае они должны учитывать ее особенности. В этой работе мы исследовали их эффективность только для задачи о неограниченном ранце.

III. Задача о неограниченном ранце

Задача о ранце - одна из наиболее изученных задач комбинаторной оптимизации, имеющая множество реальных приложений [10]. К UKP можно свести задачи управления сбором мусора, распределения ресурсов, планирования работ, составления бюджета капитальных вложений, принятия инвестиционных решений, выбора проектного решения, упаковки грузов и других.

Сформулируем задачу о неограниченном ранце в следующем виде:

1. Задан набор предметов $\{\boldsymbol{E}_i\}, i = 1, N$ для каждого из которых определены два параметра: вес $\boldsymbol{w}_i$ и стоимость $\boldsymbol{c}_i$.

2. Определен размер ранца $\boldsymbol{C}$ — максимальный вес предметов, которые в него можно положить.

3. Вариант решения задачи представляется в виде вектора $\{\boldsymbol{S}_i\}, i = 1, N$, каждый элемент которого $\boldsymbol{s}_i$ определяет количество $i-$ых предметов, помещаемых в ранец.

4. Требуется найти вектор $\{\boldsymbol{S}_i\}$, для которого выполняются следующие условия:

$$\sum_{i=1}^{N} s_i \cdot w_i \leq \boldsymbol{C}, \quad (1)$$

$$L = \sum_{i=1}^{N} s_i \cdot c_i \rightarrow max. \quad (2)$$

Если все компоненты вектора $\{\boldsymbol{S}_i\}$ целые числа, то UKP это типичная задача дискретной оптимизации.

IV. BFA и задача о ранце

Рассмотрим, как можно повысить производительность BFA с помощью определенных выше методов.

Повысить скорость генерации вариантов можно путем создания **параллельных** приложений. Для этого необходимо распределить процесс генерации между узлами вычислителя, используя OMP или MPI библиотеки. Отметим, что BFA можно запускать и в распределенной среде, поскольку он не требует постоянной коммуникации между узлами.

В **простейшем случае** генерация вариантов решения UKP может быть реализована следующим образом:

1. Рассчитаем максимальное значение разряда счетчика R. Определим:

$$q_i = (C/w_i), \forall\, i = 1, N;$$

тогда

$$R = \lfloor q_i \rfloor = max\, \lfloor C/w_i \rfloor, i = 1, N.$$

2. Реализуем $N-$ разрядный $R-$ ичный счетчик.

При этом число сгенерированных вариантов составит R^N, множество из которых не будут удовлетворять условия не переполнения ранца сразу по нескольким компонентам вектора S, если $\exists\, (C/w_i) < R, \forall\, i = 1, N$. И чем таких элементов больше, тем больше будет избыточность генератора. Оценить такую избыточность можно, используя соотношение:

$$A = R^N - \prod_{i=1}^{N} (R - q_i + 1).$$

Для уменьшения избыточности числа генерируемых вариантов, можно использовать генератор, **учитывающий особенности** задачи. В случае UKP необходимо реализовать $N-$ разрядный счетчик, для которого выполняется следующее условие:

$$s_i \le q_i, \forall\, i = 1, N;$$

т.е. максимальное значение каждого разряда ограничивается. Это позволяет устранить избыточность генератора вариантов, определенную выше. Такой вариант счетчика реализует программа COUNT_1.

Теперь рассмотрим возможность **исключения вариантов** в процессе генерации без расчета критериальной функции. Для этого можно реализовать механизм поразрядного сброса счетчика при достижении в разряде максимального значения.

Отсортируем найденные значения q_i в порядке убывания и построим $N-$ разрядный счётчик, как и в программе COUNT_1. Таким образом, младшим разрядам будут соответствовать «лёгкие» предметы, а старшим – «тяжёлые». Этот вариант счетчика реализует программа COUNT_2. Рассмотрим небольшой пример, иллюстрирующий процесс генерации.

Пусть $N = 4$ и определены значения q_i - $(2, 8, 4, 3)$. После сортировки получится - $(8, 4, 3, 2)$. Сравним работу счётчиков в программах COUNT_1 и COUNT_2. В таблице 1 представлены состояния трёх вариантов счётчиков. У первых двух счетчиков в шести строках отличий нет, но при достижении вторым разрядом максимального значения $s_2 = 3$ (строка 7) генератор вариантов решений в COUNT_2 сделает перенос в третий разряд, тогда как генератор вариантов решений в COUNT_1 продолжит стандартный счёт. При этом варианты 7 и 8 будут нарушать условие (1), поскольку $s_2 = q_2$, а $q_2 \le q_1$.

Третий вариант счётчика COUNT_3 более тонко использует тот факт, что мы сортируем значения q_i в порядке убывания. Строка 5 таблицы 1 показывает отличие третьего варианта от двух первых. Сортировка обеспечивает выполнение условия:

$$q_i \le\ q_{i-1}, \forall\, i = 2, N.$$

Тогда при достижении второго разряда счетчика значения на единицу меньшего максимума, т.е. $s_2 = q_2 - 1$, мы можем исключить вариант с $s_2 = 2, s_1 = 2$, и сразу перейти к варианту $s_3 = 1, s_2 = 0, s_1 = 0$. Генератор COUNT_3 сделает перенос в третий разряд раньше, чем первые два.

Автоматическое исключение вариантов в генераторах COUNT_2 и COUNT_3 без расчёта критериальной функции (в случае UKP это выражение 2) позволяет повысить эффективность BFA.

Таблица 1. Варианты реализации счётчиков

№	COUNT_1				COUNT_2				COUNT_3			
	s_4	s_3	s_2	s_1	s_4	s_3	s_2	s_1	s_4	s_3	s_2	s_1
1	0	0	0	1	0	0	0	1	0	0	0	1
2	0	0	0	2	0	0	0	2	0	0	0	2
3	0	0	2	0	0	0	2	0	0	0	2	0
4	0	0	2	1	0	0	2	1	0	0	2	1
5	0	0	2	2	0	0	2	2	0	0	3	0
6	0	0	3	0	0	0	3	0	0	1	0	0
7	0	0	3	1	0	1	0	0	0	1	0	1
8	0	0	3	2	0	1	0	1	0	1	0	2

V. Особенности программной реализации BFA

Выше отмечено, что одним из достоинств алгоритма перебора является простота его реализации, как в последовательном, так и в параллельном вариантах. Однако использование описанных генераторов приводит к усложнению кода. Рассмотрение всей программы не представляется интересным, поэтому приведем только фрагменты, реализующие генераторы вариантов. Параллельные варианты программ построены в виде многопоточного приложения с помощью библиотеки OpenMP.

Реализация генератора в COUNT_1.

```
1:  m = 1, i = 1, k = 1
2:  for S[N] < q[N] do
3:      for j = N to 1 do
4:          if (m == 1){S[i]+= 1;}
5:          While (S[i] == q[k]&& m < N)
6:           if(i < N){S[i] = 0; i+= 1;S[i]+= 1;k+= 1;}
7            m++
8:       if (m! = 1){ i = 1;k = 1; m = 1;}
9:       end
10:  end
```

Реализация генератора в COUNT_2.

Для проверки условия досрочного переноса в старший разряд, строки 4 и 5 приведенного выше кода заменяются на следующие команды:

```
4:      if (S[i] < q[k] && m == 1)
5:          S[i]+= 1
6:      else
7:          S[i] = q[k]
8:      if (S[N] == (q[N] − 1))
9:          S[N] = q[N]
```

Реализация генератора в COUNT_3.

```
1:  m = 1, i = 1, k = 1, sum = 0, control = 0
4:      sum = 0
5:      for p = j to 1 do
6:          sum+= S[p]
7:          if (S[j] + sum ≤ q[k])
8:          else{ j = 0; control + +;}
9:   if (control == 0){S[i]+= 1;}
10:  else {S[i] == q[k];}
16:  control = 0
```

Простейшим способом распараллеливания вычислений в многопоточных приложениях является использование функции "*omp parallel for*" из библиотеки OpenMP [11]. В представленных выше фрагментах кодов трёх генераторов формирование выходного вектора происходит от младшего разряда к старшему (Табл. 1), что требует незначительной трансформации алгоритма при его распараллеливании. Воспользуемся тем, что каждое изменение индекса в цикле "*omp parallel for*" обеспечивает запуск нового потока, и организуем разделение вычислений в потоках с помощью значения старшего разряда вектора - s_N.

Фрагмент кода для параллельной реализации четырёх поточного варианта счётчика COUNT_1 представлен ниже.

```
1: #pragma omp parallel for num_threads(4)
2:  for i = 0; i < q[N] do
3:    m = 1, i = 1, k = 1
4:    for ii = N, ii ≥ 1 do
5:      if(ii == 1) {S[ii] = i;}
6:      else {S[ii] = 0;}
7:    for p = 0; p < q[(N − 1)]; p = S[(N − 1)]
8:      for j = N to 1 do
9:      if (m == 1){ S[i]+= 1;}
10:     While (S[i] == q[k]&& m < N)
11:       if(i < (N − 1)){ S[i] = 0; i+= 1;
                  S[i]+= 1; k+= 1;}
12:         m + +
13:     if (m! = 1) { i = 1; k = 1; m = 1;}
14:   end
15: end
```

Необходимо отметить, что такое распределение нагрузки между потоками в общем случае не является сбалансированным, поскольку число вариантов вектора для разных значений старшего разряда отличается.

VI. Экспериментальная задача

Для проведения вычислительных экспериментов мы использовали UKP-тест со следующими параметрами (табл. 2):

Таблица 2. Параметры тестовой задачи

i	W_i	c_i	q_i		
			$C_1 = 40$	$C_2 = 50$	$C_3 = 60$
1	4	2	11	13	16
2	6	5	7	9	11
3	3	3	14	17	21
4	7	7	6	8	9
5	5	8	9	11	13
6	7	9	6	8	9
7	6	8	7	9	11
8	3	5	14	17	21
9	5	6	9	11	13
10	7	5	6	8	9
Число вариантов			$1{,}8 \cdot 10^9$	$1{,}9 \cdot 10^{10}$	$1{,}0 \cdot 10^{11}$

Еще одним параметром, влияющим на вычислительную сложность задачи, является вместимость рюкзака. В экспериментах этот параметр принимал следующие значения:

$$C_1 = 40,\ C_2 = 50, C_3 = 60.$$

Точное решение задачи для всех случаев было найдено с помощью динамического программирования:

$$L_1 = 66,\ L_2 = 83, L_3 = 100.$$

Отметим, что все три реализации счетчиков получили точное решение и в последовательном, и в параллельном вариантах.

VII. Результаты экспериментов

Для проведения вычислительных экспериментов использовался персональный компьютер с процессором Intel® Core™ i3 9-го поколения, количество физических ядер – 4, тактовая частота – 3,1 ГГц. Последовательные приложения запускались на одном ядре. Параллельные приложения запускались на 4-х физических ядрах, режим гипертрединга не использовался. Программирование проводилось в среде Visual Studio 2019.

Результаты вычислений показаны в таблицах 3 – 5. Параметр G_{var} это сгенерированное число вариантов, а t_{calc} – время решения задачи в секундах.

Таблица 3. Результаты работы последовательных программ

	COUNT_1		COUNT_2		COUNT_3	
C_i	G_{var}	t_{calc}	G_{var}	t_{calc}	G_{var}	t_{calc}
40	$1{,}8 \cdot 10^9$	64,8	$5{,}8 \cdot 10^8$	27,2	$4{,}3 \cdot 10^5$	9,0
50	$1{,}8 \cdot 10^{10}$	671,9	$7{,}7 \cdot 10^9$	355,2	$2{,}1 \cdot 10^6$	67,2
60	$1{,}0 \cdot 10^{11}$	3763,8	$4{,}9 \cdot 10^{10}$	2204,6	$1{,}0 \cdot 10^7$	331,2

Оценим достигнутое сокращение числа вариантов, сравнивая второй и третий счетчики с первым (табл. 4).

Таблица 4. Эффективность счетчиков

C_i	COUNT_2	COUNT_3
40	3,1	4 186
50	2,33	8 571
60	2,04	10 000

Счетчик COUNT_2 позволяет сократить генерируемое число вариантов больше чем вдвое. А вот счётчик COUNT_3 значительно эффективнее – число вариантов снижается в тысячи раз. При этом следует отметить, что более сложный счётчик - COUNT_3 за счёт нескольких проверок затрачивает большее время на генерацию одного варианта. Время решения задачи при его использовании сокращается в среднем только в 10 раз.

Посмотрим теперь, как будут работать счётчики в параллельных приложениях. Критерием оценки в данном случае будет выступать полученное по отношению к последовательной версии ускорение. В таблице 5 это параметр Accel.

Таблица 5. Результаты работы параллельных программ

	COUNT_1		COUNT_2		COUNT_3	
C_i	t_{calc}	Accel	t_{calc}	Accel	t_{calc}	Accel
40	24,79	2,61	10,25	2,65	3,51	2,57
50	251,82	2,67	132,33	2,68	24,21	2,77
60	1320,46	2,85	774,21	2,84	114,49	2,89

Все варианты реализации счетчиков показывают близкие значения получаемых ускорений, следовательно, счётчик COUNT_3 и в параллельной версии сохраняет свою эффективность.

Причин того, что приложения не достигают максимально возможного ускорения (для используемой платформы это 4), две. Первая связана с методом распределения нагрузки – использование значения старшего разряда счётчика, а вторая причина характерна для конструкции "*omp parallel for*". Максимальное ускорение она позволяет получить только если число циклов *for* делится на число параллельных потоков (в нашем случае ядер) нацело, иначе на последней стадии будет работать только одно ядро. С ростом размерности решаемых задач влияние обеих причин на эффективность вычислений будет снижаться.

VIII. Заключение

Проведенные вычислительные эксперименты подтвердили, что эффективность универсального метода решения задач дискретной оптимизации, инвариантного к условиям поиска, алгоритма полного перебора вариантов, может быть значительно повышена за счет оптимизации генератора вариантов. Отсечение неперспективных вариантов без вычисления критериальной функции позволяет повысить производительность BFA и увеличить размерность решаемых задач.

Дальнейшие исследования будут направлены на оценку возможности применения предлагаемого подхода и для других задач дискретной оптимизации – квадратичное назначение, распределение нагрузки в иерархических системах, поиск топологии беспроводных сетей и других.

Поддержка

Работа выполнена в МСЦ РАН в рамках государственного задания по теме FNEF-2022-0016.

Библиография

[1] Aye Min Thike, Sergey Lupin, "Parallel Application for Wireless Network Topology Optimisation", International Journal of Electrical, Electronics and Data Communication, vol.7, Iss.4, pp. 14-20, April 2019.

[2] Simon Bonello, [Online]. Available: https://www.chubbydeveloper.com/brute-force-algorithm.

[3] Priya Pedamkar, "Brute force algorithm for closest pair problem", [Online]. Available: https://www.educba.com/brute-force-algorithm.

[4] Abhijit Tripathy, "Brute force approach for Travelling Salesman Problem", [Online]. Available:https://iq.opengenus.org/travelling-salesman-problem-brute-force.

[5] Ms. Jahanvi Kolte, Prof. Dhaval Jha, Dr. Rajan Datt, "Strategies for Implementing the Knapsack Problem", International Journal of Science Technology and Management, Vol. No.6, Issue No. 06, June 2017, pp. 419-427.

[6] Mikhail Posypkin, Si Thu Thant Sin, "Comparative Analysis of the Efficiency of Various Dynamic Programming Algorithms for The Knapsack Problem", 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), pp. 313-316, 2-3 February 2016, St. Petersburg, Russia.

[7] O.A Balogun, M.A Dosunmu, I.O Ibidapo, "An Explanatory Comparison of Brute Force and Dynamic Programming Techniques to Solve the 0-1 Knapsack Problem", International Journal of Innovative Science and Research Technology, Vol.7, Issue 2, February 2022, pp. 660-664.

[8] Branch and Bound Algorithm, [Online]. Available: https://www.geeksforgeeks.org/branch-and-bound-algorithm.

[9] Mohammad Andri Budiman, Dian Rachmawati, "Using random search and brute force algorithm in factoring the RSA modulus", Journal of Computing and Applied Informatics (JoCAI), Vol.2, No.1, pp. 45-52, 2018.

[10] Vincent Poirriez, Nicola Yanev, Rumen Andonov, "A Hybrid Algorithm for the Unbounded Knapsack Problem", Discrete Optimization, Volume 6, 2009, pp. 110-124.

[11] OpenMP. [Online]. Available: https://en.wikipedia.org/wiki/OpenMP.

Methods for improving the efficiency of Brute-force algorithm by the example of solving an Unbounded Knapsack Problem

Aye Min Thike, S. Lupin, Min Thu Khaing

***Abstract*— A universal method for the exact solution of discrete optimization problems, invariant to search conditions, is the exhaustive search algorithm (Brute Force Algorithm, BFA). Its wide use in practice is limited by the high computational complexity determined by the dimension of the search space. This determines the interest of researchers in finding approaches to limiting the number of considered solutions without losing its accuracy. The issue of increasing the efficiency of the brute force algorithm for solving discrete optimization problems is considered. On the example of the unbounded knapsack problem (Unbounded Knapsack Problem, UKP), we show the possibility of implementing a generator that ensures the exclusion of variants that do not satisfy the conditions of the problem, without calculating the criterion function. The feasibility of generators for multi-threaded BFA applications has also been evaluated. The effectiveness of the proposed approach is confirmed for both sequential and parallel implementations. The results of computational experiments coincide with theoretical calculations. A 4-thread application with an optimized generator provides more than 30-fold acceleration of calculations. The approach under study can be applied to other discrete optimization problems as well.**

***Keywords*— brute force algorithm, parallel implementation of optimization algorithms, unbounded knapsack problem, OpenMP.**

REFERENCES

[1] Aye Min Thike, Sergey Lupin, “Parallel Application for Wireless Network Topology Optimisation”, International Journal of Electrical, Electronics and Data Communication, vol.7, Iss.4, pp. 14-20, April 2019.

[2] Simon Bonello, [Online]. Available: https://www.chubbydeveloper.com/brute-force-algorithm.

[3] Priya Pedamkar, “Brute force algorithm for closest pair problem”, [Online]. Available: https://www.educba.com/brute-force-algorithm.

[4] Abhijit Tripathy, “Brute force approach for Travelling Salesman Problem”, [Online]. Available:https://iq.open genus.org/travelling-salesman-problem-brute-force.

[5] Ms. Jahanvi Kolte, Prof. Dhaval Jha, Dr. Rajan Datt, “Strategies for Implementing the Knapsack Problem”, International Journal of Science Technology and Management, Vol. No.6, Issue No. 06, June 2017, pp. 419-427.

[6] Mikhail Posypkin, Si Thu Thant Sin, “Comparative Analysis of the Efficiency of Various Dynamic Programming Algorithms for The Knapsack Problem”, 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), pp. 313-316, 2-3 February 2016, St. Petersburg, Russia.

[7] O.A Balogun, M.A Dosunmu, I.O Ibidapo, “An Explanatory Comparison of Brute Force and Dynamic Programming Techniques to Solve the 0-1 Knapsack Problem”, International Journal of Innovative Science and Research Technology, Vol.7, Issue 2, February 2022, pp. 660-664.

[8] Branch and Bound Algorithm, [Online]. Available: https://www.geeksforgeeks.org/branch-and-bound-algorithm.

[9] Mohammad Andri Budiman, Dian Rachmawati, “Using random search and brute force algorithm in factoring the RSA modulus”, Journal of Computing and Applied Informatics (JoCAI), Vol.2, No.1, pp. 45-52, 2018.

[10] Vincent Poirriez, Nicola Yanev, Rumen Andonov, “A Hybrid Algorithm for the Unbounded Knapsack Problem”, Discrete Optimization, Volume 6, 2009, pp. 110-124.

[11] OpenMP. [Online]. Available: https://en.wikipedia.org/wiki/ OpenMP.