О применении локального поиска с чередующимися окрестностями для настройки параметров решателей

Устюгов В.Н., Институт математики им. С.Л. Соболева СО РАН (Омский филиал)

Исследование выполнено за счет гранта Российского научного фонда № 22-71-10015, https://rscf.ru/en/project/22-71-10015/

Мотивация

У пакетов прикладных программ (далее *решателей*) для решения задач частично-целочисленного линейного программирования (ЧЦЛП) существуют параметры влияющие на ход решения задач и предназначенные для задания человеком. Подбор этих параметров слишком трудозатратная задача, поэтому цель этого исследования - найти лучший способ автоматически оптимизировать параметры решателей. Очевидно, что не существует универсального набора параметров, и для каждого класса задач оптимальный набор свой.

Постановка

Оптимизация параметров проводилась для задачи построения расписания для многоядерного процессора с учётом взаимного влияния работ (*Еремеев, Сахно, 2023*). Рассматриваемая задача состоит в том, чтобы запланировать выполнение работ на ядрах процессора с учетом их замедления при совместном выполнении. Ограничение на порядок выполнения этих работ задается как частичный порядок на множестве работ. Такая постановка не подразумевает, что предшествующие работы как-либо влияют на последующие, например, посредством кеша или температуры процессора. **Цель** состоит в том, чтобы свести к минимуму время завершения последней работы.

Подход к решению. Variable neighborhood search (VNS)

Алгоритм VNS

Шаг 1. Выбрать окрестности $N_k, k = 1, \ldots, k_{\max}$, и начальную точку x.

Шаг 2. Повторять, пока не выполнен критерий остановки.

2.1. $k \leftarrow 1$.

2.2. Повторять, пока $k \leqslant k_{\max}$:

(a) случайно выбрать точку $x' \in N_k(x)$;

(b) применить локальный спуск с начальной точки x', не меняя координат, по которым x и x' совпадают. Полученный локальный оптимум обозначается x'';

(c) если $f(x'') < f(x)$, то полагается $x \leftarrow x'', k \leftarrow 1$, иначе $k \leftarrow k + 1$.

Подходы к решению. Adaptive capping

В своей реализации алгоритма VNS я применил технику усечения, предложенную в статье о ParamILS (Hutter et al, 2009). Цель этой техники в том, чтобы заранее останавливать расчёт *очевидно* неперспективных точек исследуемого пространства.

Механизм следующий: производится расчёт целевой метрики для набора задач на очередной точке пространства параметров. Если получается так, что задачи дорешались не до конца, а целевая метрика уже “хуже”, чем в текущей точке, то алгоритм не досчитывает до конца все индивидуальные задачи, а завершает расчёт для этой точки, и начинает расчёт для следующей.

Подходы к решению. Adaptive capping

Для времени выполнения: считается суммарное время, если на шаге расчёта для новой точки оно выше, чем для всех задач в текущей, то происходит отказ от решения и перемещение в другую точку.

Для разрыва двойственности: аналогично, только вместо суммирования вычисляется средний разрыв для рассчитанных задач, с гипотезой, что для всех остальных (ещё не решённых в этой точке задач), разрыв равняется нулю.

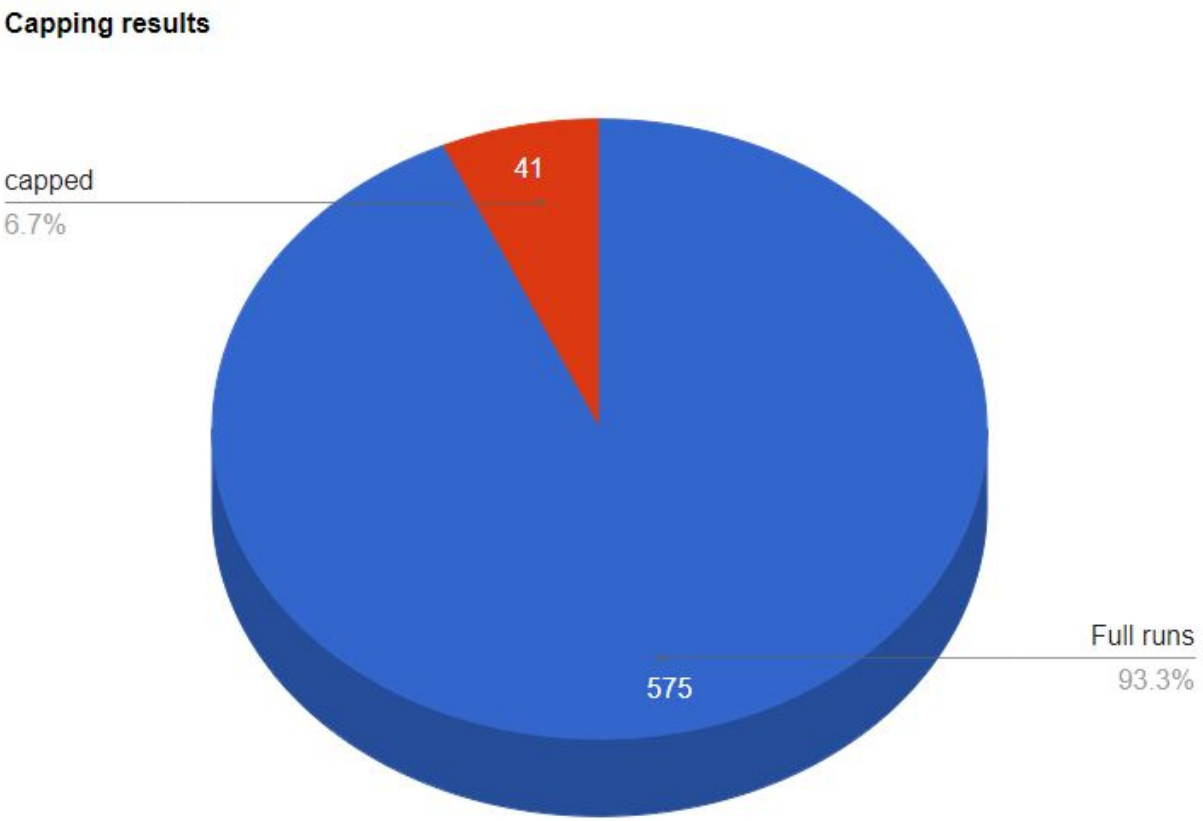
Результаты вычислительных экспериментов

Далее представлено суммарное время на всех задачах. Эксперименты проводились на сервере AMD EPYC ОФ ИМ СО РАН. Стоит обратить внимание на то, что

1. Настройка проводилась на 10% от всего набора задач
2. Настройка проводилась только на задачах с 4 работами

	4 jobs	7 jobs
Параметры по-умолчанию	2,572 с	2,592 с
Параметры после настройки	2,543 с	2,576 с

Результаты. Capping



Дальнейшие планы

- Провести аналогичные эксперименты для всех размерностей
- Использовать кросс-валидацию (скользящий контроль)

Спасибо за внимание!

Математическая постановка задачи. Имеется множество работ $J = \{1, \ldots, n\}$ и m ядер процессора. Работы выполняются непрерывно и не меняют ядро в процессе выполнения. На одном ядре не может выполняться более одной работы.

Для каждой работы $j \in J$ известно количество единиц времени s_j, необходимое ей для полного выполнения в идеальных условиях (т.е. при условии, что вместе с ней не выполняется других работ).

Введем определения. *Конфигурация* — это набор работ, выполняющихся одновременно на разных ядрах с учетом частичного порядка на множестве работ (конфигурация не может содержать пару работ, в которой одна из работ должна быть выполнена раньше другой в соответствии с частичным порядком с учетом транзитивности) и ограничений на количество ядер. Множество всех конфигураций обозначим C. Положим, что в нулевой конфигурации ни одна работа не выполняется. На множестве C также задан частичный порядок, составленный на основе частичного порядка на множестве работ. Т.е. конфигурация c_1 предшествует конфигурации c_2 в том случае, если хотя бы одна из работ конфигурации c_1 должна выполниться раньше хотя бы одной работы из конфигурации c_2 . Заметим, что число элементов в множестве C ограничено сверху $\sum_{i=0}^{m} C_n^i$.

Скоростью выполнения работы j в конфигурации $c \in C$ будем называть отношение времени выполнения работы j в идеальных условиях ко времени полного выполнения работы j, если бы j все это время выполнялась в конфигурации c. Скорость работы зависит от конфигурации, в которой она выполняется. На протяжении каждой конфигурации скорость выполнения всех работ считается постоянной.

Итак, для каждой конфигурации $c \in C$ известно, из каких работ она состоит. Для каждой работы j в конфигурации c известна скорость ее выполнения v_{jc}.

Необходимо составить такое расписание выполнения работ на ядрах процессора, что общее время завершения работ $C_{\max}$ минимально.

{'threads': 8, 'presolve': -1, 'gomorypasses': 5, 'method': 2, 'minrelnodes': 0, 'mipfocus': 0}