Задачи распределения в реальном времени

Сергей Николенко

Теория экономических механизмов — ИТМО, весна 2008

Outline

Суть задачи

- Есть процессор (один), на вход ему потихоньку поступают задачи.
- Вопрос в том, чтобы последовательно выбирать, над какой задачей из доступных работать в настоящий момент.
- Очевидно, что все задачи выполнить не получится; надо выбирать при поступлении задачи, что с ней делать.

Суть задачи

- Давайте пока безо всяких механизмов.
- Просто поступают задачи, их надо распределить.
- Задача — это пара (e, d): время исполнения и дедлайн.
- Если задача поступила в момент t_0, мы можем получить e единиц «дохода», выделив e слотов процессорного времени внутри интервала $t_0 \leq t \leq t_0 + d$.

Суть задачи

- Т.е. мы хотим максимизировать суммарное время полезной работы процессора.
- Для этого нужен онлайн-алгоритм, который будет реагировать на поступающие задачи.
- Мы будем сравнивать его с оптимальным ясновидящим алгоритмом, который всё знает заранее и может найти глобальный оптимум.
- Алгоритм r-оптимален, если он достигнет доли как минимум r от результата оптимального алгоритма.

Пример и верхняя оценка

- Первая идея верхней оценки такая: давайте возьмём большую задачу S_i длиной L_i.
- А потом будем давать кучу мелких задач, каждая стоит ϵ, начинаются следующая во время дедлайна предыдущей.
- Как только алгоритм откажется от большой задачи, мелкие закончатся, и он получит всего ϵ.
- Если не откажется, получит большую задачу. Большая задача называется *приманкой*.

Пример и верхняя оценка

- Обозначения: время делится на эпохи.
- В каждой эпохе противник сначала делает большую задачу T_0 длины $t_0 = 1$.
- Потом, за ϵ до конца большой задачи T_i длиной t_i запускает задачу T_{i+1} длиной t_{i+1}.
- Ни одна эпоха не продолжается дальше T_m для некоторого m.
- Все задачи, кроме T_m, приманки.

Пример и верхняя оценка

- Во-первых, игрок ни разу не оставит приманку ради мелкой задачи, потому что она ему даст за всю эпоху не больше малого ϵ.
- Во-вторых, это всё значит, что в течение одной эпохи игрок получит либо одну задачу T_i, $i < m$, либо задачу T_m.
- А оптимальный алгоритм сможет навыполнять все мелкие задачи, потому что в таком случае перекрывающиеся большие задачи ему не помешают. Т.е. он фактически берёт

Пример и верхняя оценка

- Давайте теперь введём такие последовательности времён:

$$t_{i+1} = (ct_i - \sum_{j=0}^{i} t_j), \ c \text{ константа.}$$

- Тогда, если игрок только T_i берёт, то он получит t_i, а противник — $\sum_{j=0}^{i+1} t_j$ (T_{i+1} успеет получиться).
- Отношение получается

$$\frac{t_i}{\sum_{j=0}^{i+1} t_j} = \frac{t_i}{ct_i - \sum_{j=0}^{i} t_j + \sum_{j=0}^{i} t_j} = \frac{1}{c}.$$

Пример и верхняя оценка

- А если игрок выполнил T_m, то у него t_m, а у противника $\sum_{j=0}^{m} t_j$.
- Надо только выбрать c и m так, чтобы $\frac{t_m}{\sum_{j=0}^{m} t_j}$ тоже было не больше $\frac{1}{c}$.
- Т.е. надо найти минимальное c такое, что функция $t : \mathbb{N} \to \mathbb{N}$, заданная рекуррентным соотношением

$$t_0 = 1, \quad t_{i+1} = (ct_i - \sum_{j=0}^{i} t_j),$$

удовлетворяла бы условию

$$\exists m \geq 0 : \frac{t_m}{\sum_{j=0}^{m} t_j} \leq \frac{1}{c}.$$

Пример и верхняя оценка

- Тут есть простор для комбинаторных рассуждений.

Упражнение. Докажите, что $\exists m \geq 0 : \frac{t_m}{\sum_{j=0}^{m} t_j} \leq \frac{1}{c}$ эквивалентно $\exists l \geq 0 : t_{l+1} \leq t_l$.

Упражнение. Докажите, что рекуррентное условие эквивалентно

$$t_0 = 1, \quad t_1 = c - 1, \quad t_{i+2} = c(t_{i+1} - t_i).$$

Пример и верхняя оценка

- А если к последнему уравнению применить стандартную теорию рекуррентно заданных последовательностей, получится, что нужное условие выполняется для $c < 4$.

Упражнение. Изучите эту теорию и примените её.

1/4-оптимальный алгоритм TD_1

- Мы тут выяснили, что лучше $\frac{1}{4}$ не бывает.
- Сейчас построим алгоритм, который достигает $\frac{1}{4}$-оптимальности.
- Сначала рассмотрим ситуацию, когда у задач нету laxity — т.е. если начинать, то начинать прямо сейчас.
- Это позволит нам резко упростить алгоритм.

1/4-оптимальный алгоритм TD_1

- Задача описывается как $T_i = (a_i, c_i, d_i, v_i)$:
  - r_i — release, когда пришла;
  - c_i — computation, сколько надо времени;
  - d_i — deadline, когда должно быть сделано;
  - v_i — value, сколько дадут.
- Обозначим $l_i = d_i - c_i$ самое позднее время начала.

1/4-оптимальный алгоритм TD_1

- Введём понятие *интервала* в t. Интервал — это промежуток времени $[t_b, t_e)$, где есть занятый подпромежуток $[t_b, t_f]$, а за ним (если надо) подпромежуток простоя $[t_f, t_e)$, где $t_b \leq t$ — это когда система начала работать, t_f — это когда система перейдёт (ожидается, что перейдёт) снова в нерабочее состояние, потому что задача закончится, и

$$t_e = \max(t_f, \max(d_{disc})),$$

где d_{disc} — дедлайны задач, от которых придётся отказаться на протяжении $[t_b, t_f]$.

- Интервал *замкнутый*, если на нём завершили задачу, *открытый* в противном случае.

1/4-оптимальный алгоритм TD_1

- Рассмотрим интервал Δ. Его размер зависит от задач. Он растёт, поглощая новые открытые интервалы, пока не станет замкнутым.
- Рассмотрим последовательность поступающих задач

$$(T_{a_1}, T_{a_2}, \ldots, T_{a_n}) \text{ и соответствующие интервалы}$$

$$(\Delta_{a_1}, \Delta_{a_2}, \ldots, \Delta_{a_n}), \text{ где } \Delta_n = \Delta.$$

- Обозначим

$$(T_1, T_2, \ldots, T_k)$$

задачи, которые реально выполняются системой, причём T_i прерывает предыдущую задачу, и

$$(\Delta_1, \Delta_2, \ldots, \Delta_k), \ -$$

соответствующие интервалы. Последовательности интервалов возрастают и там, и здесь.

1/4-оптимальный алгоритм TD_1

- Опишем алгоритм TD_1. Сначала предположим, что у всех нету laxity, т.е. надо сразу решать.
- Алгоритм такой: когда приходит новая задача T_{next},
  - проапдейтить Δ_{run} (текущий интервал);
  - если $v_{run} < \Delta_{run}/4$, то $T_{run} = T_{next}$.
- Т.е. если новая стоимость вчетверо превышает текущий размер интервала, взять новую задачу.

1/4-оптимальный алгоритм TD_1

- Первая лемма: $v_k > \frac{\Delta_k}{2}$.
- Для $k = 1$ $v_1 = \Delta_1 > \Delta_1/2$.
- Поскольку T_i прекращается T_{i+1}, $v_i < \Delta_{i+1}/4$, $\Delta_{i+1} < \Delta_i + v_{i+1}$.

Упражнение. Докажите, что тогда $2v_{i+1} > \Delta_{i+1}$ (по индукции).

1/4-оптимальный алгоритм TD_1

- Теперь докажем, что на каждом интервале получается не менее четверти от ясновидящего алгоритма.
- Первый случай: $T_{a_n} = T_k$, т.е. после T_k никаких задач не приходило. Тогда просто $v_k > \frac{\Delta_k}{2} = \frac{\Delta}{2} > \frac{\Delta}{4}$.
- Второй случай: после T_k ещё приходили задачи, но были отброшены. Раз были отброшены, значит, $v_k \geq \frac{\Delta_{a_n}}{4} = \frac{\Delta}{4}$.

1/4-оптимальный алгоритм TD_1

- Что делать, когда есть laxities?
- Нужно использовать очередь Q, в которой лежат ожидающие старта задачи, отсортированные по l_i (времени самого позднего старта).
- Приходит новая задача, помещается в Q. Если система свободна, она выполняет первую задачу из Q.
- Если несвободна, решение принимается, когда наступает самое раннее l_i из имеющихся.

1/4-оптимальный алгоритм TD_1

- То есть получается, что первая задача интервала может иметь laxity в момент начала, а остальные уже нет.
- Если алгоритм выполнит первую задачу с ненулевым laxity, а кое-что не выполнит из-за этого (а остальное с нулевым laxity ведь), то ясновидящий алгоритм, может, смог бы задержать первую, выполнить срочные, вернуться к первой...
- Поэтому введём переменную *pl* (potential loss) — значение первой задачи в каждом интервале.

1/4-оптимальный алгоритм TD_1

- Алгоритм теперь такой: когда приходит новая задача, она помещается в Q.
- Когда система освобождается и Q непуста, $T_{run} = \text{dequeue}(Q)$, $pl = v_{run}$.
- Когда система работает и звучит сигнал (наступает l_i задачи из Q):
  - $T_{next} = \text{dequeue}(Q)$; $\text{update}(\Delta_{run})$.
  - Если $v_{run} < (\Delta_{run} + pl)/4$, то $T_{run} = T_{next}$.

1/4-оптимальный алгоритм TD_1

- Доказательство похоже на предыдущее. Рассмотрим такие же последовательности задач и интервалов T_i, T_{a_i}, Δ_i, Δ_{a_i}.
- Сначала такая же индукция.

Упражнение. Докажите, что $v_k \geq \frac{\Delta_k + pl}{2}$.

- А затем те же два случая, и так же гарантируется, что $v_k \geq \frac{\Delta_{a_n} + pl}{4}$.

1/4-оптимальный алгоритм TD_1

- Можно это дело немножко обобщить на случай, когда количество одновременных задач ограничено, но мы лучше перейдём к делу, то есть к дизайну механизмов.

Outline

Зачем тут дизайн механизмов

- Вот интересный пример. Рассмотрим три задачи:
  - $r_1 = 0,\ d_1 = 0.9,\ c_1 = 0.9,\ v_1 = 0.9;$
  - $r_2 = 0.5,\ d_2 = 5.5,\ c_2 = 4,\ v_2 = 4;$
  - $r_3 = 4.8,\ d_3 = 17,\ c_3 = 12.2,\ v_3 = 12.2.$
- Что сделает TD_1?

Зачем тут дизайн механизмов

- Он выполнит T_1, потом начнёт выполнять T_2.
- А в момент 4.8, когда надо будет решать про T_3, он подсчитает, что

$$\frac{t_e - t_b + pl}{4} = \frac{17 - 0.9 + 4}{4} > 4 = v_2,$$

и начнёт выполнять третью задачу.

Зачем тут дизайн механизмов

- Зачем дизайн механизмов? А вот зачем.
- Пусть тот, кто даёт вторую задачу, соврёт, что его дедлайн $\widehat{d}_2 = 4.7$.
- Тогда алгоритм рассмотрит вторую задачу в момент 0.7 и предпочтёт её первой, т.к. $\frac{4.7-0.0+1}{4} > 0.9 = v_1$.
- И вторая задача успеет завершиться до начала третьей.

Формулировка

- У нас есть центр и N агентов, центр не знает N, агент каждый владеет одной задачей i.
- Характеристики задачи — это тип агента θ_i.
- Во время r_i агент i узнаёт свой тип θ_i и, начиная с этого момента, может предлагать задачу процессору.
- Он это делает, объявляя $\hat{\theta}_i = (\hat{r}_i, \hat{d}_i, \hat{c}_i, \hat{v}_i)$, а затем функция $g : \Theta \to \mathcal{O}$ выбирает исход, т.е. расписание.

Формулировка

- Важно: мы не будет отдавать задачу обратно агенту до его объявленного дедлайна $\hat{d}_i$, даже если посчитали раньше
- Это нам поможет для правдивости потом.
- Полезность для каждого агента

$$u_i(g(\hat{\theta}), \theta_i) = v_i \mu(e_i(\hat{\theta}, d_i) \geq c_i) \mu(\hat{d}_i \leq d_i) - p_i(\hat{\theta}),$$

где μ — индикатор своего аргумента, p_i — выплата, которую должен сделать агент, e_i — сколько процессор на данное время работал над задачей i.

Формулировка

- Т.е. агенты квазилинейные.
- Ещё ограничение: агент не может дать длину меньше настоящей, т.к. центр заметит. И не может дать задачу центру до настоящего r_i, т.к. сам её ещё не знает.
- Т.е. агент даёт $\hat{\theta}_i = (\hat{r}_i, \hat{d}_i, \hat{c}_i, \hat{v}_i)$, где $\hat{r}_i \geq r_i$, $\hat{c}_i \geq c_i$.

Формулировка

- Ещё фишка — у нас теперь разные c_i и v_i.
- Поэтому надо знать верхнюю оценку на то, каким бывает отношение $\frac{v_i}{c_i} = \rho$.
- Отношение называется плотностью; пусть $\rho_{\min} = 1$ (wlog), а $\rho_{\max} = k$.
- Наш механизм будет $((1 + \sqrt{k})^2 + 1)$-оптимальным.

Механизм

- Теперь о механизме. Он не даёт предпочтений имеющейся работе (TD_1 хотел, чтобы новая задача была аж вчетверо лучше).
- Он просто исполняет работу с максимальным приоритетом

$$\hat{v}_i + \sqrt{k} e_i(\hat{\theta}, t)\rho_{\min}.$$

- Когда чья-нибудь работа выполняется, мы с агента берём сумму по правилу «второй цены»: берём минимальное v, которое он мог бы заявить так, чтобы его работа всё же выполнилась.

Механизм

- Тогда сразу, как в Викри-аукционах или VCG, автоматически получится рациональность и правдивость относительно стоимостей v_i.

Упражнение. Докажите это.

Механизм

- Нужно только понять, почему он правдив относительно трёх других параметров: r_i, c_i и d_i.
- Во-первых, улучшать (уменьшать) r_i и c_i мы уже запретили.
- Во-вторых, улучшать (увеличивать) d_i тоже бессмысленно: тогда агенту отдадут работу, когда ему уже поздно.
- Именно для этого нужно было отдавать работу не сразу после выполнения.

Упражнение. Придумайте пример, в котором было бы выгодно отодвигать дедлайн, если бы работу выдавали сразу по выполнению.

Механизм

- Нужно доказать, почему агенту невыгодно ухудшать параметры работы, делать её строже.
- Мы подробно доказывать не будем — много технических деталей, которые мы уже разбирали в других ситуациях.
- Идея такая.

Механизм

- Если увеличивать длину работы, единственный эффект от этого — задержка выполнения или вообще отказ от выполнения работы (у неё приоритет ухудшается).
- Если приближать дедлайн, можно добиться, что работу раньше сделают, но от этого радости агенту в нашей постановке нет. Но больше шанс, что от работы откажутся.
- Не так очевидно, почему нехорошо отодвигать r_i (время объявления работы центру).
- Но тоже можно доказать.

Механизм

- Осталось доказать про нужную степень оптимальности.
- Это делается примерно так же, как в анализе TD_1.
- Разобьём время на интервалы $(t_i^o, t_i^c]$, где во время t_i^c завершается работа i, а во время t_i^o завершается t_{i+1}^o $(t_1^o = 0)$.
- Пусть время t_i^b — это первое время, когда на интервале i процессор работает.

Механизм

- Тогда практически такой же индукцией, как раньше, можно доказать требуемую оценку на интервалы.

Упражнение. Докажите по индукции, что

$$t_i^c - t_i^b \geq \left(1 + \frac{1}{\sqrt{k}}\right) v_i.$$

Механизм

- А затем можно доказать, что в каждом интервале мы не отбрасываем слишком дорогие задачи.

Упражнение. Для каждого интервала I_i и задачи j, которая была тогда отброшена,

$$v_j \leq (1 + \sqrt{k})v_i.$$

Механизм

- Дальнейший анализ совсем уж опустим, но в результате получится требуемая оптимальность. :)
- В общем, итог такой: дизайн механизмов помог разработать такую систему, в которой агенты заинтересованы в выполнении своих задач, но врать им при этом незачем.

Спасибо за внимание!

- Lecture notes и слайды будут появляться на моей homepage:

  `http://logic.pdmi.ras.ru/~sergey/index.php?page=teaching`

- Присылайте любые замечания, решения упражнений, новые численные примеры и прочее по адресам: `sergey@logic.pdmi.ras.ru`, `snikolenko@gmail.com`
- Заходите в ЖЖ  **smartnik**.