جزوه جلسه ۴

(انتهای فصل ۱ و ابتدای فصل ۴)

سید سقاعده هاشمی

۱۹ دی ۱۳۹۶

یادآوری

در این بخش به طور خلاصه مسئله‌ای که قصد حل آن را داریم مرور می‌کنیم.

داشتیم مسئله پوشش مجموعه‌ای [۱] را حل می‌کردیم. در این مسئله ما یک مجموعه جامع $E = \{e_1, \ldots, e_n\}$ و زیرمجموعه‌های $S_1, \ldots, S_m$ از آن را داریم. می‌خواهیم تعدادی از این زیرمجموعه‌ها را انتخاب کنیم به طوری که اجتماع آن‌ها برابر E شود. اما برای انتخاب مجموعه S_j باید هزینه w_j را پرداخت کنیم. بنابراین می‌خواهیم جوابمان کمترین هزینه را داشته باشد. (در این نوشته برای سادگی دنبال کردن مبحث، همواره اندیس i را برای اعضای مجموعه E و اندیس j را برای مجموعه‌های $S_1, \ldots, S_m$ به کار می‌بریم. بنابراین همواره $1 \leq i \leq n$ و $1 \leq j \leq m$ برقرار است)

این مسئله را به صورت یک مسئله برنامه‌ریزی صحیح [۲] به این صورت نوشتیم: به ازای هر مجموعه S_j یک پارامتر x_j تعریف کردیم که فقط می‌تواند یکی از دو مقدار ۰ و ۱ را اختیار کند که ۰ به معنای عدم انتخاب زیرمجموعه S_j و ۱ به معنای انتخاب آن است. با معرفی این دسته پارامترها، مسئله برنامه‌ریزی صحیح را این گونه نوشتیم:

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{m} w_j x_j \\
\text{subject to} \quad & \sum_{j: e_i \in S_j} x_j \geq 1 \quad \forall i \in \{1, \ldots, n\} \\
& x_j \in \{0, 1\} \qquad\quad \forall j \in \{1, \ldots, m\}
\end{aligned} \tag{۱}$$

هزینه جواب بهینه این مسئله را OPT نامیدیم.

حل یک مسئله برنامه‌ریزی صحیح کار بسیار مشکلی است (NP-hard است). به همین خاطر سعی کردیم پاسخ این مسئله را با مسئله ساده‌تری تخمین بزنیم. با آرام‌سازی [۳] شرط ۰ یا ۱ بودن x_j ها و کنار گذاشتن محدودیت صحیح بودن آن‌ها به مسئله‌ی برنامه‌ریزی خطی [۴]

[۱] set cover
[۲] integer programming
[۳] relaxation
[۴] linear programming

رسیدیم.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{m} w_j x_j^* \\
\text{subject to} \quad & \sum_{j: e_i \in S_j} x_j^* \geq 1 \quad \forall i \in \{1, \ldots, n\} \\
& 0 \leq x_j^* \qquad\qquad \forall j \in \{1, \ldots, m\}
\end{aligned} \tag{۲}
$$

هزینه پاسخ این مسئله را Z_{LP}^* نامیدیم. دیدیم که می‌توان با انتخاب یک قاعده گردکردن [۵] خوب، از بهترین جواب مسئله برنامه‌ریزی خطی به یک جواب برای مسئله برنامه‌ریزی صحیح رسید و رابطه‌ای بین هزینه آن جواب گرد شده Z_{LP}^* برقرار کرد.

ابتدا با یک قاعده گرد کردن قطعی [۶] به یک الگوریتم f - تقریب رسیدیم که $f := \max_{1 \leq i \leq n} |\{j : e_i \in S_j\}|$. سپس مسئله دوگان [۷] مسئله برنامه‌ریزی خطی را نوشتیم و سعی کردیم آن را حل کرده و با استفاده از جواب آن، جواب مسئله اولیه [۸] را بسازیم. تا به اینجا با داشتن x_j^* ها x_j های متناظر را به صورت قطعی حساب کردیم. حال چه می‌شود اگر x_j ها را به صورت تصادفی بسازیم؟ آیا به تخمین بهتری می‌رسیم؟

۱ مقدمه

در این جلسه می‌خواهیم پاسخ مسئله برنامه‌ریزی خطی را با یک الگوریتم گردکردن غیرقطعی [۹] به پاسخ مسئله برنامه‌ریزی صحیح تبدیل کنیم. در جلسات قبل دیدیم که اگر پاسخ مسئله برنامه‌ریزی خطی $\mathbf{x}^* = [x_1^* \ldots x_m^*]^T$ باشد می‌توانیم پاسخ مسئله برنامه‌ریزی صحیح را به صورت زیر بسازیم:

$$
x_j = x_j^{(3)} := \begin{cases} 1 & \text{با احتمال } x_j^* \\ 0 & \text{با احتمال } 1 - x_j^* \end{cases} \tag{۳}
$$

در این صورت امید هزینه پاسخ مسئله برنامه‌ریزی صحیح به صورت زیر است:

$$
\mathbf{E}\left[\sum_{j=1}^{m} w_j x_j\right] = \sum_{j=1}^{m} w_j \mathbf{E}[x_j] = \sum_{j=1}^{m} w_j x_j^* = Z_{LP}^* \leq OPT \tag{۴}
$$

بنابراین امید هزینه‌ای که این الگوریتم ایجاد می‌کند از جواب مسئله برنامه‌ریزی صحیح کمتر است!!! پس اشکالی در کار است. اشکالی که در این الگوریتم وجود دارد این است که ممکن است جوابی که برای برنامه‌ریزی صحیح می‌سازیم طوری باشد که اجتماع مجموعه‌های انتخاب شده E نشود.

برای حل این مشکل خوب است احتمال صحیح نبودن جواب را بررسی کنیم. برای این کار ابتدا احتمال پوشیده نشدن یک $e_i \in E$ را

[۵] rounding
[۶] deterministic
[۷] dual
[۸] primal
[۹] nondeterministic

محاسبه می‌کنیم.

$$P[\bar{e_i}^{(3)}] := P[\text{ احتمال پوشیده نشدن } e_i \text{ وقتی } x_j \text{ ها با قاعده 3 ساخته شوند }]$$
$$= P[\text{ انتخاب نشدن هیچ یک از زیرمجموعه‌هایی که شامل } e_i \text{ است }] \quad (۵)$$
$$= \Pi_{j:e_i \in S_j}(1 - x_j^*)$$

می‌توانیم حد بالایی برای $(1 - x_j^*)$ به دست آوریم که محاسبه حاصل‌ضرب عبارت پایانی تساوی‌های ۵ را ساده کند. اگر بسط تیلور تابع e^x را برای $x = -x_j^*$ حول ۰ بنویسیم داریم

$$e^{-x_j^*} = e^0 - x_j^* e^0 + x_j^{*2} e^0 - \cdots \geq e^0 - x_j^* e^0 = 1 - x_j^* \quad (۶)$$

پس

$$P[\bar{e_i}^{(3)}] \leq \Pi_{j:e_i \in S_j} e^{-x_j^*} = e^{-\sum_{j:e_i \in S_j} x_j^*} \quad (۷)$$

بنابر شرط برنامه‌ریزی خطی $\sum_{j:e_i \in S_j} x_j^* \geq 1$ پس

$$P[\bar{e_i}^{(3)}] \leq e^{-1} \quad (۸)$$

حال که این احتمال را محاسبه کردیم، بیایید احتمال اشتباه بودن جواب را محاسبه کنیم.

$$P[\bar{E}^{(3)}] := P[\text{ حداقل یک } e_i \text{ پوشیده نشده باشد }] = P[\cup_{1 \leq i \leq n} \bar{e_i}^{(3)}] \leq \sum_{1 \leq i \leq n} P[\bar{e_i}^{(3)}] = ne^{-1} \quad (۹)$$

اما عملا برای $n > 2$ به نامساوی بدیهی $P[\bar{E}^{(3)}] \leq 1 < ne^{-1}$ می‌رسیم. اما می‌توان با تکنیک ساده و جالبی این مشکل را حل کرد.

۲ اصلاح اولیه

ما دوست داریم $P[\bar{E}]$ را تا حد امکان کم کنیم. برای این کار می‌توانیم احتمال انتخاب شدن S_j ها (که برابر احتمال ۱ شدن x_j است) را از x_j^* بیشتر کنیم. پس قاعده ساختن x_j ها را به این شکل تغییر می‌دهیم:

$$x_j = x_j^{(10,k)} := \begin{cases} 1 & \text{با احتمال } \min\{1, kx_j^*\} \\ 0 & \text{با احتمال } 1 - \min\{1, kx_j^*\} \end{cases} \quad (۱۰)$$

حال مجددا احتمال پوشیده نشدن یک e_i را بررسی می‌کنیم:

$$P[\bar{e_i}^{(10,k)}] := P[\text{ احتمال پوشیده نشدن } e_i \text{ وقتی } x_j \text{ ها با قاعده 10 ساخته شوند }] = \Pi_{j:e_i \in S_j}(1 - \min\{1, kx_j^*\}) \quad (۱۱)$$

اگر $kx_j^* \geq 1$ باشد این احتمال ۰ می‌شود. پس این دو حالت را مجزا بررسی می‌کنیم. ابتدا حالتی را بررسی می‌کنیم که $kx_j^* < 1$ باشد.

$$P[\bar{e_i}^{(10,k)}] = \Pi_{j:e_i \in S_j}(1 - kx_j^*) \leq \Pi_{j:e_i \in S_j} e^{-kx_j^*} = e^{-k\sum_{j:e_i \in S_j} x_j^*}$$
$$= \left(e^{-\sum_{j:e_i \in S_j} x_j^*}\right)^k \leq (e^{-1})^k = e^{-k} \quad (۱۲)$$

اگر $kx_j^* \geq 1$ نیز نامساوی $P[\bar{e_i}^{(10,k)}] = 0 \leq e^{-k}$ برقرار است.

حال احتمال اشتباه بودن پاسخ الگوریتم در صورتی که از قاعده ۱۰ استفاده کنیم را محاسبه می‌کنیم:

$$\begin{aligned} P[\bar{E}^{(10,k)}] &:= P[\text{ احتمال اشتباه بودن جواب وقتی } x_j \text{ ها با قاعده 10 ساخته شوند }] \\ &= P[\cup_{1\leq i\leq n}\bar{e_i}^{(10,k)}] \leq \sum_{1\leq i\leq n} P[\bar{e_i}^{(10,k)}] \leq ne^{-k} \end{aligned} \tag{۱۳}$$

پس اگر بخواهیم پاسخ ما با احتمال $\frac{1}{n}$ اشتباه باشد باید

$$ne^{-k} \leq \frac{1}{n} \Rightarrow e^{-k} \leq \frac{1}{n^2} \Rightarrow -k \leq -2\log n \Rightarrow k \geq 2\log n \tag{۱۴}$$

در اصل می‌توانیم این الگوریتم را برای هر دقت دلخواهی استفاده کنیم. مثلا به ازای یک $\epsilon > 0$ برای این که $P[\bar{E}^{(10,k)}] < \epsilon$ باید

$$ne^{-k} < \epsilon \Rightarrow -k < \log\epsilon - \log n \Rightarrow k > \log n - \log\epsilon \tag{۱۵}$$

که چون $\log\epsilon$ وقتی $\epsilon \to 0$ به سرعت به ∞ میل می‌کند، این الگوریتم k را به شدت زیاد می‌کند. اما افزایش k در کنار افزایش احتمال صحت جواب، احتمال انتخاب S_j ها را نیز افزایش می‌دهد پس شهودا باید جوابی که تولید می‌کند هزینه زیادی داشته باشد. برای دقیق کردن این ادعا، امید و واریانس هزینه با این الگوریتم را حساب می‌کنیم.

$$\begin{aligned} \mathbf{E}\left[\sum_{j=1}^{m} w_j x_j\right] &= \mathbf{E}\left[\sum_{j=1}^{m} w_j x_j^{(10,k)}\right] = \sum_{j=1}^{m} w_j \mathbf{E}[x_j^{(10,k)}] = \sum_{j=1}^{m} w_j \min(1, kx_j^*) \\ &\leq \sum_{j=1}^{m} w_j k x_j^* = k \times Z_{LP}^* \leq k \times OPT \end{aligned} \tag{۱۶}$$

$$Var\left(\sum_{j=1}^{m} w_j x_j\right) = \sum_{j=1}^{m} Var(w_j x_j^{(10,k)}) = \sum_{j=1}^{m} w_j^2 \min(1, kx_j^*)(1-\min(1,kx_j^*)) \tag{۱۷}$$

پس اگر $k \to \infty$ امید هزینه به سمت انتخاب تمام مجموعه‌ها میل می‌کند و واریانس نیز به ۰ میل می‌کند.

یک راه جایگزین برای قاعده ۱۰ وجود دارد که یک برتری احتمالاتی نسبت به این روش دارد. می‌توان به جای ضرب کردن x_j^* در k، یک سکه که احتمال شیر آمدن آن x_j^* است را k بار انداخت. اگر حداقل یک بار شیر آمد x_j را ۱ و در غیر این صورت ۰ بگذاریم. این قاعده را می‌توان به صورت زیر فرمال کرد:

$$x_j = x_j{}^{(18,k)} := \begin{cases} 1 & \text{با احتمال } 1-(1-x_j^*)^k \\ 0 & \text{با احتمال } (1-x_j^*)^k \end{cases} \tag{۱۸}$$

ابتدا احتمال صحیح نبودن جواب این قاعده را بررسی می‌کنیم:

$$\begin{aligned} P[\bar{e_i}^{(18,k)}] &:= P[\text{ احتمال پوشیده نشدن } e_i \text{ وقتی } x_j \text{ ها با قاعده 18 ساخته شوند }] \\ &= \Pi_{j:e_i\in S_j}(1-x_j^*)^k = [\Pi_{j:e_i\in S_j}(1-x_j^*)]^k \\ &\leq [\Pi_{j:e_i\in S_j} e^{-x_j^*}]^k = [e^{-\sum_{j:e_i\in S_j} x_j^*}]^k = e^{-k} \end{aligned} \tag{۱۹}$$

حال احتمال صحیح نبودن پاسخ وقتی گرد کردن را با قاعده ۱۸ انجام دهیم، محاسبه می‌کنیم:

$$\begin{aligned} P[\bar{E}^{(18,k)}] &:= P[\text{ احتمال اشتباه بودن جواب وقتی } x_j \text{ ها با قاعده 18 ساخته شوند }] \\ &= P[\cup_{1\leq i\leq n}\bar{e}_i^{(18,k)}] \leq \sum_{1\leq i\leq n} P[\bar{e}_i^{(18,k)}] \leq ne^{-k} \end{aligned} \tag{۲۰}$$

پس این قاعده از نظر احتمال صحت جواب با قاعده ۱۰ مشابه است. حال به بررسی هزینه‌ای که این قاعده تولید می‌کند می‌پردازیم. در ادامه محاسبات از یک نامساوی استفاده کردیم که آن را در اینجا ذکر و اثبات می‌کنیم. برای x های مثبت نزدیک به صفر نامساوی زیر برقرار است

$$1-(1-x)^k \leq kx \tag{۲۱}$$

برای اثبات این نامساوی، سری تیلور تابع $f(x)=1-(1-x)^k$ را حول $x=0$ می‌نویسیم.

$$f(x)=f(0)+xf'(0)+x^2f''(0)+O(x^3)=0+kx-k(k-1)x^2+O(x^3) \tag{۲۲}$$

اگر x به اندازه کافی کوچک باشد

$$f(x)\approx kx-k(k-1)x^2 \tag{۲۳}$$

و چون $k(k-1)x^2>0$ پس

$$1-(1-x)^k=f(x)\leq kx \tag{۲۴}$$

پس نامساوی مورد نظرمان را ثابت کردیم. حال به ادامه محاسبات مربوط به قاعده ۱۸ می‌پردازیم.

$$\begin{aligned} \mathbf{E}\left[\sum_{j=1}^m w_jx_j\right] &= \mathbf{E}\left[\sum_{j=1}^m w_jx_j^{(18,k)}\right]=\sum_{j=1}^m w_j\mathbf{E}[x_j^{(18,k)}]=\sum_{j=1}^m w_j(1-(1-x_j^*)^k) \\ &\leq \sum_{j=1}^m w_jkx_j^* = kZ_{LP}^* \leq k\times OPT \end{aligned} \tag{۲۵}$$

$$Var\left(\sum_{j=1}^m w_jx_j\right)=\sum_{j=1}^m Var(w_jx_j^{(18,k)})=\sum_{j=1}^m w_j^2(1-x_j^*)^k(1-(1-x_j^*)^k) \tag{۲۶}$$

پس اگر $k\to\infty$ امید هزینه، به انتخاب تمام مجموعه‌ها میل می‌کند و واریانس هزینه نیز به ۰. پس در حد عملکرد قاعده ۱۰ و ۱۸ یکی است اما در عمل، قاعده ۱۸ امید بهتر اما واریانس بدتری دارد. تا پایان این بخش به اثبات این دو ادعا می‌پردازیم.

ابتدا سعی می‌کنیم ادعایمان درباره مقایسه امید هزینه این دو قاعده را ثابت کنیم:

$$\begin{aligned} 1-(1-x_j^*)^k \leq kx_j^* &\Rightarrow 1-(1-x_j^*)^k\leq \min(1,kx_j^*) \Rightarrow w_j(1-(1-x_j^*)^k)\leq w_j\min(1,kx_j^*) \\ &\Rightarrow \sum_{j=1}^m w_j(1-(1-x_j^*)^k)\leq\sum_{j=1}^m w_j\min(1,kx_j^*) \\ &\Rightarrow \mathbf{E}\left[\sum_{j=1}^m w_jx_j^{(18,k)}\right]\leq \mathbf{E}\left[\sum_{j=1}^m w_jx_j^{(10,k)}\right] \end{aligned} \tag{۲۷}$$

پس امید هزینه‌ای که قاعده ۱۸ ایجاد می‌کند از قاعده ۱۰ بهتر است.

برای اثبات زیاد بودن واریانس هزینه قاعده ۱۸ نسبت به قاعده ۱۰ از یک ایده زیرکانه استفاده می‌کنیم. می‌دانیم اگر $0 \le a \le b \le c \le d \le 1$ و $a + d = b + c$ برقرار باشد $ad \le bc$ است. می‌خواهیم ثابت کنیم

$$0 \le 1 - \min(1, kx_j) \le (1 - x_j)^k \le 1 - (1 - x_j)^k \le \min(1, kx_j) \le 1 \tag{۲۸}$$

و نتیجه بگیریم

$$\begin{aligned} &\min(1, kx_j)(1 - \min(1, kx_j)) \le (1 - x_j)^k(1 - (1 - x_j)^k) \\ &\Rightarrow Var(x_j^{(10,k)}) \le Var(x_j^{(18,k)}) \\ &\Rightarrow Var\left(\sum_{j=1}^{m} w_j x_j^{(10,k)}\right) \le Var\left(\sum_{j=1}^{m} w_j x_j^{(18,k)}\right) \end{aligned} \tag{۲۹}$$

می‌دانیم $1 - (1 - x_j)^k \le kx_j$ است. چون $0 \le x_j \le 1$ پس $1 - (1 - x_j)^k \le 1$ و در نتیجه $1 - (1 - x_j)^k \le \min(1, kx_j)$.

همچنین می‌دانیم $(1 - x_j)^k \ge 1 - kx_j$ است. از طرفی $(1 - x_j)^k \ge 0$ پس $1 - \min(1, kx_j) \le (1 - x_j)^k$.

حال تنها اثبات نامساوی $(1 - x_j)^k \le 1 - (1 - x_j)^k$ باقی مانده. اگر $(1 - x_j) = 1$ باشد، این نامساوی برقرار است. اما اگر $0 \le (1 - x_j) < 1$ حتما یک k' وجود دارد که برای هر $k \ge k'$ نامساوی $(1 - x_j)^k \le \frac{1}{2}$ برقرار باشد. پس برای هر $k \ge k'$ نامساوی $(1 - x_j)^k \le 1 - (1 - x_j)^k$ صحیح است.

پس ثابت کردیم از یک k به بعد، واریانس هزینه قاعده ۱۸ از قاعده ۱۰ بیشتر است.

۳ اصلاح دوم

در قسمت قبل توانستیم الگوریتمی برای گردکردن ارائه کنیم که احتمال تولید پاسخ اشتباه آن $\frac{1}{n}$ باشد. الگوریتم‌هایی که به صورت احتمالی جواب صحیحی ارائه می‌کنند لزوما در اولین اجرا به جواب صحیح نمی‌رسند پس لازم است صحت خروجی اولین اجرا را آزمایش کنیم و اگر نادرست بود الگوریتم را مجددا اجرا کنیم و پاسخ جدیدش را آزمایش کنیم و باز اگر اشتباه بود الگوریتم را مجددا اجرا کنیم و ... . اولین سوالی که به ذهن خطور می‌کند این است که «باید این الگوریتم را چند بار اجرا کنیم؟». نمی‌توان به صورت قطعی به این سوال پاسخ داد اما می‌توان حساب کرد امید ریاضی تعداد دفعات اجرا تا زمانی به جواب صحیح برسیم چیست. در این مسئله چون احتمال تولید جواب صحیح $1 - \frac{1}{n}$ است پس امید ریاضی تعداد دفعات لازم

$$\mathbf{E}[\text{تعداد دفعات لازم}] = \frac{1}{1 - \frac{1}{n}} = \frac{n}{n-1} \le 2 \tag{۳۰}$$

است.

پس الگوریتم نهایی ما این گونه شد:

«مسئله برنامه‌ریزی خطی را حل می‌کنیم. الگوریتم گردکردن غیر قطعی را اجرا می‌کنیم و صحت پاسخش را آزمایش می‌کنیم. این قدر الگوریتم گردکردن غیرقطعی را تکرار می‌کنیم تا نهایتا به یک جواب صحیح برسیم و در آخر همان جواب را ارائه می‌کنیم»

پس پاسخ الگوریتم ما قطعا صحیح است اما زمان اجرا و کیفیت آن(تفاوت هزینه پاسخ ما و OPT) تصادفی است. چون زمان اجرای الگوریتم گردکردن غیرقطعی و آزمایش صحت جواب ارائه شده توسط آن، چند جمله‌ای است و امید تعداد اجرای لازم در این مسئله، حداکثر ۲ است، پس امید زمان اجرای الگوریتم، چند جمله‌ای است.

حال به بررسی کیفیت الگوریتم می‌پردازیم. برای بررسی کیفیت الگوریتم، الگوریتم را برای k دلخواه بررسی می‌کنیم. (نه فقط آن k که $P[\bar{E}^{(18,k)}] = \frac{1}{n}$) امید هزینه را به صورت زیر می‌نویسیم:

$$\mathbf{E}\left[\sum_{j=1}^{m} w_j x_j \middle| E^{(18,k)}\right] \tag{۳۱}$$

که $E^{(18,k)}$ رویداد پوشیده شدن تمام E با استفاده از قاعده گردکردن تصادفی ۱۸ است. درواقع می‌خواهیم امید هزینه را به شرط صحیح بودن جواب محاسبه کنیم. برای سادگی نوشتار، در ادامه محاسبات به جای $\sum_{j=1}^{m} w_j x_j$ عبارت $cost$ را می‌نویسیم.

$$\begin{aligned} &\mathbf{E}[cost] = \mathbf{E}\left[cost \middle| E^{(18,k)}\right] P\left[E^{(18,k)}\right] + \mathbf{E}\left[cost \middle| \bar{E}^{(18,k)}\right] P\left[\bar{E}^{(18,k)}\right] \\ &\Rightarrow \mathbf{E}[cost] \geq \mathbf{E}\left[cost \middle| E^{(18,k)}\right] P\left[E^{(18,k)}\right] \\ &\Rightarrow \mathbf{E}\left[cost \middle| E^{(18,k)}\right] \leq \frac{\mathbf{E}[cost]}{P\left[E^{(18,k)}\right]} \leq \frac{k \times OPT}{1 - ne^{-k}} \end{aligned} \tag{۳۲}$$

اگر $k = 2\log n$ باشد:

$$\mathbf{E}\left[cost \middle| E^{(18,2\log n)}\right] \leq \frac{2\log n \times OPT}{1 - \frac{1}{n}} \leq \frac{2\log n \times OPT}{\frac{1}{2}} = 4\log n \times OPT \tag{۳۳}$$

پس توانستیم ثابت کنیم امید هزینه این الگوریتم غیرقطعی، $4\log n$ - تقریب است.

پس توانستیم کیفیت و زمان اجرای الگوریتم غیرقطعی‌مان را تحلیل کنیم.

۴ شروع فصل ۴

در این فصل می‌خواهیم تکنیک گرد کردن قطعی را روی چند مسئله به کار ببریم و قدرت آن را بررسی کنیم.

اولین مسئله‌ای که قصد دست و پنجه نرم کردن با آن را داریم مسئله «زمان‌بندی کارها»ست. در این مسئله، ما فرض می‌کنیم یک ماشین داریم که به مشتری‌ها خدمات محاسباتی ارائه می‌دهد. مشتری‌ها برای این ماشین «کار» ارسال می‌کنند. مثلا این ماشین در دقیقه r_j یک کار دریافت می‌کند که اجرای آن p_j دقیقه زمان نیاز دارد. این ماشین ممکن است در حین انجام یک کار، کار دیگری دریافت کند. در این حالت مجاز نیست کاری که شروع کرده را کنار بگذارد و بعدا آن را ادامه دهد. به عبارت دیگر، وقتی این ماشین یک کار را شروع می‌کند بدون توقف تا پایان آن را انجام می‌دهد. (این حالت مسئله را حالت «غیرانحصاری [۱۰] » می‌نامند) ممکن است این ماشین تصمیم بگیرد بعد از پایان فلان کار، کار دیگری را شروع نکند و منتظر دریافت یک کار مشخص بماند. (دقت کنید که این ماشین قبل از این که یک کار به دستش برسد می‌داند چه زمانی آن کار را دریافت می‌کند و می‌داند آن کار چه قدر زمان نیاز دارد) در نهایت این ماشین قصد دارد ترتیب اجرای این کارها را طوری تنظیم کند که مجموع زمان اجرای کارها کمینه شود. به طور دقیق‌تر اگر زمان پایان کار j را $c_j^{(N)}$ بنامیم، این ماشین می‌خواهد $\sum_{j=1}^{n} c_j^{(N)}$ را کمینه کند. البته ممکن است معیار دیگری را انتخاب کنیم. مثلا مجموع زمان معطل شدن مشتری‌ها ($\sum_{j=1}^{n} c_j^{(N)} - r_j$) یا تاخیر ماشین (

[۱۰] nonpreemptive

$\sum_{j=1}^{n} c_j^{(N)} - r_j - p_j$) مثال‌های رایجی هستند. این مسئله کاربرد بسیار ارزشمندی در زمان‌بندی کارهای CPU توسط سیستم‌عامل دارد و بسیاری از کلمات و اصطلاحات آن از این حوزه نشأت گرفته. البته مسئله‌ای که سیستم‌عامل‌ها حل می‌کنند دشوارتر است چون زمان‌های r_j را نمی‌دانند و در نتیجه باید آن‌ها را نیز تخمین بزنند.

در تلاش اول سعی می‌کنیم با آرام‌سازی شرط غیرانحصاری بودن ماشین، مسئله ساده‌تری بسازیم و راهی برای حل آن و تبدیل پاسخ مسئله جدید به پاسخی برای مسئله اصلی پیدا کنیم. ابتدا مسئله آرام‌سازی شده را به صورت فرمال می‌نویسیم.

« n کار در زمان‌های $r_1, \ldots, r_n$ به ماشین ارسال می‌شوند که اجرای آن‌ها $p_1, \ldots, p_n$ زمان نیاز دارد. می‌توانیم اجرای یک کار را متوقف کنیم و بعدا آن کار را از همان جایی که قبلا آن را متوقف کردیم ادامه دهیم. مثلا ممکن است روی کاری که ۷ دقیقه طول می‌کشد ابتدا ۲ دقیقه کار کنیم. سپس آن را متوقف کنیم و به سراغ کارهای دیگر برویم. بعد از مدتی مجددا به سراغ همین کار برگردیم و ۴ دقیقه دیگر روی آن وقت بگذاریم و مجددا آن را متوقف کنیم و به سراغ کارهای دیگر برویم و بعد از انجام چند کار دیگر، به همین کار برگردیم. ۱ دقیقه دیگر روی آن وقت بگذاریم و آن را تمام کنیم. زمان پایان کار j در این مسئله را $c_j^{(P)}$ می‌نامیم. می‌خواهیم ترتیبی برای اجرای کارها تنظیم کنیم که $\sum_{j=1}^{n} c_j^{(P)}$ کمینه شود.»

ادعا می‌کنیم این مسئله با الگوریتم زیر حل می‌شود:

«اولین کاری که دریافت کردی را شروع کن. سپس هر وقت کار جدیدی دریافت کردی، زمان پایان آن و پایان باقی‌مانده کار جدید را مقایسه کن و آن کاری که زودتر تمام می‌شود را انجام بده. وقتی کاری را تمام کردی نیز از بین کارهای موجود کاری را شروع کن که زمان باقی‌مانده از آن از سایر کارها کمتر باشد.»

حال اثبات می‌کنیم این الگوریتم، بهترین جواب را تولید می‌کند. برای این کار از برهان خلف استفاده می‌کنیم. فرض می‌کنیم چینش دیگری برای انجام کارها وجود دارد که هزینه مورد نظر ما را کمتر می‌کند. فرض می‌کنیم دنباله $c_1^{(P)}, \ldots, c_n^{(P)}$ توسط الگوریتم ما تولید شده و دنباله $c'_1, \ldots, c'_n$ همان دنباله فرض خلف است. اولین جایی که این دو دنباله با هم تفاوت دارند را در نظر می‌گیریم. مثلا برای یک k داریم

$$
\begin{aligned}
c'_1 &= c_1^{(P)} \\
&\vdots \\
c'_{k-1} &= c_{k-1}^{(P)} \\
c'_k &\neq c_k^{(P)}
\end{aligned}
\tag{۳۴}
$$

بنابر نحوه انتخاب کارها در الگوریتم ما، کاری که ما در نوبت k ام قرار دادیم زودتر از کار k ام الگوریتم دیگر تمام می‌شود پس $c_k^{(P)} < c'_k$ است. فرض می‌کنیم کاری که ما در مرحله k ام قرار دادیم، در چینش الگوریتم دیگر، در مرحله k' ام تمام می‌شود. ادعا می‌کنیم اگر جای کارهای k ام و k' ام در چینش فرض خلف را عوض کنیم به هزینه کمتری می‌رسیم. برای اثبات این ادعا فرض می‌کنیم چینش جدیدی که از روی چینش $c'_1, \ldots, c'_n$ ساختیم به صورت $c''_1, \ldots, c''_n$ است. بنابر نحوه ساختن چینش جدید داریم:

$$
\begin{aligned}
c'_1 &= c''_1 = c_1^{(P)} \\
&\vdots \\
c'_{k-1} &= c''_{k-1} = c_{k-1}^{(P)} \\
c'_k &> c''_k = c_k^{(P)}
\end{aligned}
\tag{۳۵}
$$

چون جای کار k ام و کار k' ام در چینش فرض خلف را عوض کردیم پس برای تمام $k < j < k'$ تساوی $c'_j - c''_j = c'_k - c''_k$ برقرار است. به عبارت ساده‌تر، کارهای بین k و k' به اندازه تفاضل $c'_k - c''_k$ زودتر شروع شده و در نتیجه زودتر تمام می‌شوند. اما $c'_{k'} = c''_{k'}$ زیرا مجموعه کارهای ۱ تا k' تغییری نکرده و در نتیجه در زمان یکسانی به پایان می‌رسند. پس از روی چینش فرض خلف چینش دیگری به دست آوردیم که هزینه کمتری دارد. بنابراین به تناقض رسیدیم.

پس ثابت کردیم الگوریتم معرفی شده کمترین هزینه را تولید می‌کند. (دقت کنید که یک کار در الگوریتم ما و طی فرایند اثبات چندین اندیس متفاوت می‌گیرد. مثلا فلان کار در ورودی اندیس j دارد و r_j, p_j برای آن داده می‌شود. سپس در چینش الگوریتم ما، کارها مجددا بر حسب ترتیب زمان پایان مرتب شده و اندیس جدیدی می‌گیرند. مثلا ممکن است کاری که در ورودی با اندیس ۱ به ما داده شده، در چینش ارائه شده توسط الگوریتم ما، پنجمین کاری باشد که تمام می‌شود بنابراین داده‌های مربوط به آن در تحلیل خروجی الگوریتم با اندیس ۵ مشخص می‌شوند $r_5, p_5, c_5^{(P)}$)

حال باید سعی کنیم پاسخ مسئله انحصاری را به مسئله غیرانحصاری تبدیل کنیم. در مسئله انحصاری ممکن است بعد از پایان کار سوم، کار چهارم هنوز نرسیده باشد اما کار پنجم رسیده باشد. ما کار پنجم را شروع می‌کنیم و تا زمانی که کار چهارم برسد آن را ادامه می‌دهیم. به محض این که کار چهارم رسید، کار پنجم را متوقف می‌کنیم و کار چهارم را انجام می‌دهیم تا تمام شود. اما در مسئله غیرانحصاری نمی‌توانیم این گونه عمل کنیم. یک راه ساده برای تبدیل پاسخ مسئله انحصاری به پاسخ مسئله غیرانحصاری این چنین است:

«مسئله انحصاری را حل کنیم و پاسخ $c_1^{(P)}, \ldots, c_n^{(P)}$ را بگیریم. حال ترتیب اجرای کارها در مسئله غیرانحصاری را معادل ترتیب تمام شدن کارها در مسئله انحصاری بگذاریم. مثلا کاری که در دقیقه $c_1^{(P)}$ در پاسخ مسئله انحصاری تمام می‌شود را به عنوان اولین کار اجرا کنیم. (ممکن است این کار هنوز به به ماشین نرسیده باشد. در این حالت ماشین صبر می‌کند تا این کار به دستش برسد.) بعد از پایان این کار، کاری که در مسئله انحصاری در دقیقه $c_2^{(P)}$ اجرا می‌شود را اجرا می‌کنیم. (مجددا ممکن است این کار هنوز به دست ماشین نرسیده باشد) و به همین ترتیب ادامه می‌دهیم.»

سوالی که پاسخ آن برای ما اهمیت دارد این است که «پاسخی که این الگوریتم گردکردن می‌سازد چه قدر به پاسخ بهینه نزدیک است؟»

چون در مسئله انحصاری امکان شکستن کارها و انجام تکه‌تکه آن‌ها را داریم قطعا هزینه بهترین پاسخ مسئله غیرانحصاری از بهترین پاسخ مسئله انحصاری بیشتر است.

$$\sum_{j=1}^{m} c_j^{(P)} \leq OPT \tag{۳۶}$$

که OPT هزینه بهترین پاسخ مسئله غیرانحصاری است. پس اگر ثابت کنیم

$$\sum_{j=1}^{m} c_j^{(N)} \leq k \sum_{j=1}^{m} c_j^{(P)} \tag{۳۷}$$

ثابت می‌شود

$$\sum_{j=1}^{m} c_j^{(N)} \leq k \times OPT \tag{۳۸}$$

برای این کار نامساوی زیر را اثبات می‌کنیم.

$$c_j^{(N)} \leq \max_{1 \leq k \leq j} r_k + \sum_{k=1}^{j} p_j \leq 2c_j^{(P)} \tag{۳۹}$$

نامساوی سمت چپ برقرار است زیرا طرف راست آن بیان می‌کند که تا لحظه دریافت تمام کارهای ۱ تا j صبر کنیم و سپس شروع به انجام کارهای ۱ تا j کنیم. بنابراین به وضوح زمان پایان کارهای ۱ تا j با این روش از $c_j^{(N)}$ بیشتر است. برای اثبات نامساوی سمت راست، دو نامساوی زیر را ثابت می‌کنیم:

$$\max_{1 \leq k \leq j} r_k \leq c_j^{(P)} \tag{۴۰}$$

$$\sum_{k=1}^{j} p_j \leq c_j^{(P)} \tag{۴۱}$$

نامساوی اول به این دلیل درست است که قطعا زمان پایان کار j ام از زمان دریافت تمام کارهای ۱ تا j بیشتر است. (چون باید آن کارها اول دریافت شوند و بعد اجرا شوند) معنای نامساوی دوم هم این است که اگر از همان لحظه اول، بدون توجه به زمان دریافت کارها، کارهای ۱ تا j را انجام دهیم، قطعا زمان پایان این کارها از $c_j^{(P)}$ کمتر می‌شود. پس نامساوی ۳۹ ثابت شد. حال روی $1 \leq j \leq n$ از دو طرف نامساوی ۳۹ سیگما می‌گیریم.

$$\sum_{j=1}^{m} c_j^{(N)} \leq 2 \sum_{j=1}^{m} c_j^{(P)} \leq 2 \times OPT \tag{۴۲}$$

پس ثابت کردیم که این الگوریتم، یک الگوریتم ۲-تقریب برای مسئله غیرانحصاری است. نکته قابل توجه این است که اگر تابعی که قصد کمینه کردن آن را داریم را عوض کنیم ممکن است الگوریتم تقریبی خوبی به دست نیاوریم. مثلا اگر به جای $\sum_{j=1}^{m} c_j^{(N)}$ قصد کمینه کردن $\sum_{j=1}^{m} c_j^{(N)} - r_j$ را داشته باشیم، ممکن است مسئله حاصل الگوریتمی با تقریب خوب نداشته باشد.