ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Сборник трудов
1964 г. Института математики СО АН СССР Выпуск II

ПОСТАНОВКА ЗАДАЧИ АНАЛИЗА СЕТЕВОГО ГРАФИКА

К.А. Багриновский, И.Б. Рабинович

Сетевым графиком принято называть ориентированный граф без контуров, дуги которого имеют одну или несколько числовых характеристик.

Сетевой график обычно ассоциируется с описанием некоторого производственного процесса во времени, поэтому принята некоторая установившаяся терминология: вершины сетевого графика называют событиями, дуги графа – работами.

Числовые характеристики дуги непосредственно связаны с продолжительностью выполнения соответствующей работы. Если считать процесс полностью детерминированным, то каждой дуге u_{ij}, соединяющей вершины i и j графа, ставится в соответствие число $\tau_{ij} \geq 0$, имеющее смысл точной длительности работы. Но так как в большинстве случаев процесс нельзя считать строго детерминированным, то задаются некоторые возможные сроки выполнения работы и указывается распределение вероятностей. Чаще всего такая информация задаётся с помощью трех временных оценок: так называемых "оптимистического", "наиболее вероятного", и "пессимистического" времени выполнения работы. При этом обычно соответствующие вероятности полагаются равными $\frac{1}{6}, \frac{2}{3}, \frac{1}{6}$. При обработке этих оценок можно говорить о математическом ожидании времени выполнения работы и считать в первом приближении так же, как и в детерминированных процессах, что каждой дуге u_{ij} поставлено в соответствие одно число τ_{ij}.

В настоящей статье мы остановимся на простейшей задаче анализа сетевого графика, имея в виду графы, описывающие детерминированные процессы. Однако основные проблемы и трудности простейшего анализа остаются в силе и для стохастических процессов.

Задачу анализа сетевого графика сформулируем следующим образом:

Дано множество U всех дуг графа, каждая дуга (работа) u_{ij} задаётся тройкой чисел (i , j , τ_{ij}).

Здесь i - код номера вершины, из которой дуга исходит (код события - начала работы);

j - код номера вершины, в которую дуга входит (код события - конца работы);

$\tau_{ij} \geq 0$ - продолжительность работы (этому числу можно придать смысл длины дуги ij).

Как известно, путем в графе называется последовательность дуг, таких, что началом каждой следующей дуги является конец предыдущей. В сетевом графике длиной пути считается сумма длин всех составляющих его дуг.

Требуется построить алгоритм, реализуемый на современных ЭВМ, который позволил бы при указанных данных решить следующие частные задачи:

I. Для каждого события найти наименьшее возможное время его свершения, иными словами, для каждой вершины e в графе найти путь наибольшей длины, начинающийся в одном из входов графа и заканчивающийся в вершине e . Длину этого пути мы будем обозначать через T_e . В частности, путь наибольшей длины из всех путей, начинающихся в одном из входов графа и заканчивающихся в одном из выходов графа, называется критическим путем данного сетевого графика. Особая важность отыскания этого пути хорошо известна.

2. Для каждого события найти наиболее позднее возможное время его свершения, т.е. то предельное время, после которого всякая задержка свершения события повлечет за собой отсрочку выполнения всего комплекса работ.

Нетрудно заметить, что эта задача равносильна следующей: найти путь наибольшей длины, начинающийся в вершине e графа и заканчивающийся в вершине, которая является концом критического пути. Обозначим длину такого наибольшего пути через T_e' . Тогда искомое наиболее позднее возможное время свершения события. $\overline{T}_e = T_{крит} - T_e'$,

где через $T_{крит}$ обозначается длина критического пути.

При помощи найденных значений T_e и $\overline{T}_e$ можно для каждого события e определить так называемый "резерв времени":

$$r_e = \overline{T}_e - T_e .$$

Из определения критического пути немедленно вытекает, что для всех вершин, лежащих на критическом пути,

$$T_e + T_e' = T_{крит},$$

откуда

$$T_e = T_e' \qquad \text{и} \qquad r_e = 0$$

3. Провести частичное упорядочение событий (вершин графа) таким образом, чтобы подготовить необходимую информацию для вычерчивания сетевого графика. Эту задачу можно решить при помощи нахождения порядка каждой вершины графа или при помощи построения в том или ином виде матрицы смежности.

Приведенные выше три частные задачи являются, по нашему мнению, необходимыми элементами простейшей задачи анализа сетевого графика в том смысле, что всякий более или менее общий алгоритм анализа должен давать решения этих частных задач. Укажем, кроме того, на некоторые существенные трудности решения задач анализа и вытекающие из их преодоления требования ко всему алгоритму решения.

I. В точной постановке задачи анализа предполагается, что анализируемый граф не содержит контуров (замкнутых путей). Однако вследствие ошибок при формировании исходного множества работ (дуг), ошибок при кодировке на ЭВМ и целого ряда других причин, граф, определяемый множеством заданных работ, содержит контуры. Совершенно ясно, что решение задачи анализа в этом случае становится бессмысленным. Устранение же ошибок вручную, которое при небольшом числе работ легко проводится путем вычерчивания соответствующего сетевого графика, при большом числе дуг (порядка нескольких тысяч) становится совершенно невозможным. Поэтому к основным требованиям, которые должны быть предъявлены к алгоритмам, реализуемым на ЭВМ, следует отнести необходимость автоматического отыскания контуров и указания вершин, входящих в контур, для исправления допущенных ошибок. Эта процедура может быть осуществлена либо в виде неко-

торой предварительной проверки исходных данных, либо в ходе самого решения задачи анализа. Вряд ли можно считать возможным отыскание всех контуров данного графа, однако нахождение по крайней мере одного, если он существует, должно быть обеспечено.

2. Большие размеры исходного множества работ, вообще говоря, требуют использовать, кроме оперативных запоминающих устройств ЭВМ, также и внешнюю память. Отсюда вытекает еще одно важное требование к алгоритму анализа, заключающееся в том, что необходимые для его реализации обращения к внешней памяти не увеличивали бы существенно время работы ЭВМ. Точнее, непригодным следует считать такой алгоритм, для реализации которого число обращений к внешней памяти сравнимо с количеством работ в исходном множестве U, иными словами, информация для своей обработки должна вводиться в оперативную память достаточно крупными массивами. Это очевидное требование, его следует всегда иметь в виду при перенесении алгоритмов (разработанных для малого количества работ) на большие массивы. Ясно, что при небольших размерах множества анализ сетевого графика на ЭВМ не представляется затруднительным. Если множество работ таково, что вся исходная информация помещается в оперативной памяти, то можно указать количество приёмов, позволяющих решить поставленные задачи, включая отыскание контуров, за малое время работы машины.

В настоящее время существует несколько методов анализа сетевого графика на ЭВМ. Некоторые методы имеют более или менее общий характер, т.е. позволяют найти решение всех трех частных задач, указанных выше. Как правило, эти методы либо используют выигрышные особенности конкретных электронных машин, либо позволяют обработать ограниченное количество информации.

Некоторыми другими методами можно решать отдельные частные задачи, но для решения более общей проблемы анализа эти методы нуждаются в существенном дополнении.

При этом отдельные методы отличаются друг от друга тем, что они в равной мере удовлетворяют вышеизложенным требованиям.

Далее, попытаемся описать некоторые из известных нам алгоритмов и дать им оценку.

Метод анализа сетевого графика, предложенный при разработке системы "PERT", заключается в последовательном выполнении 10 операций, имеющих номера от 01 до 10. Исходные данные (множество работ U) вводятся в машину с перфокарт.

Операция 01 состоит в записи исходной информации на магнитную ленту № 1.

Операция 02 обрабатывает исходную информацию, находящуюся на входной магнитной ленте № 1, приводя её к стандартному виду, описанному выше, т.е. каждая работа характеризуется тремя числами i, j, τ_{ij},
где i – код события (начала работ),
j – код события (конца работ),
τ_{ij} – продолжительность работы.

Операция 03 состоит в упорядочении множества работ u по коду события i (начала работы). После выполнения операции множество U записывается на первой вспомогательной ленте так, что все работы, имеющие один и тот же код i, образуют группу работ, в которой коды конца работы являются элементами множества Γi (множества событий, непосредственно следующих за событием i). Здесь Γ означает отображение, реализуемое данным графом.

Операция 04 состоит в упорядочении множества работ U по коду события j (конец работы). После выполнения операции 04 множество U записывается на второй вспомогательной ленте так, что все работы, имеющие один и тот же код j, образуют одну группу, в которой коды начала работы образуют множество $\Gamma^{-1} j$.

Операция 05 осуществляет объединение информации, имеющейся на обеих вспомогательных лентах, так, что событие e со своими множествами Γe и $\Gamma^{-1} e$ записывается на магнитную ленту (четвертую по счету), называемую первоначальным накопителем событий.

Операция 06 служит для корректировки первоначального накопителя событий по ходу выполнения плана работ, в указанной нами постановке задачи она не представляется существенной. После корректировки первоначального накопителя событий на магнитной ленте (пятой по счету) образуется и записывается накопитель событий. Структура информации накопителя событий аналогична структуре первоначального накопителя событий.

Операция 07 состоит в частичном упорядочении множества событий E по отношению следования, определяемого графом. В результате выполнения этой операции события – элементы множества E – располагаются на ленте в такой последовательности, что e_2 не ставится раньше любого события e_1, если $e_2 \prec e_1$.

Упорядоченное таким образом множество событий записывается в накопитель событий. Операция 07 выполняется при помощи конечного (вообще говоря, большого) числа перемоток шести быстродействующих магнитных лент. Совершенно ясно, что вышеупомянутое частичное упорядочение возможно лишь в том случае, если рассматриваемый сетевой график является графом без контуров. Однако операция 07 может быть построена так, что в процессе её выполнения контуры могут быть выявлены и образующие их последовательности работ выданы для исправления ошибок.

Операция 08 состоит в вычислении времени T_e и $\overline{T}_e$ для каждого события e и резерва времени τ_e.

Благодаря предварительному упорядочению, сделанному по операции 07, вычисление времен T_e и $\overline{T}_e$ выполняется при помощи одного просмотра накопителя событий, а также последовательного перехода от предыдущих событий к последующим (для вычисления T_e) и от последующих к предыдущим (для вычисления $\overline{T}_e$).

Подсчет T_e и $\overline{T}_e$ производится по итерационным формулам:

$$T_e = 0$$ для входов графа;

$$T_e = max(T_i + T_{ie};\ T_e)\ (i \in \Gamma_e^{-1})$$ для остальных вершин;

$$T'_e = T_{крит}$$ для конца критического пути;

$$\overline{T}_e = min(\overline{T}_j - \tau_{ej};\ \overline{T}_e)\ (j \in \Gamma_e)$$ для остальных вершин.

Операция 09 служит для приведения событий к первоначальному виду (см. описание операции 05), что делается для последующей корректировки графика выполнения плана работ (операция 06).

Операция 10 – операция выдачи результатов. Выдача результатов производится по принципу возрастания резерва времени: первыми выдаются события, имеющие $\tau_e = 0$, затем события, имеющие малый резерв времени и т.д.

Описанный алгоритм имеет достаточно общий характер, так как позволяет найти решения указанных выше трех частных задач анализа. Критический путь представлен как множество событий, имеющих резерв времени $\tau_e = 0$; для всех остальных событий резерв времени вычислен по операции 08; благодаря проведенному частичному упорядочению можно будет вычертить сетевой график.

Данный алгоритм использует существенно выигрышные специфические особенности электронной машины, на которой он реали-

зуется. К этим особенностям, как нам кажется, следует прежде всего отнести возможность переноса исходных данных с перфокарт на магнитную ленту непосредственно и наличие большого количества быстродействующих магнитных запоминающих устройств.

Другой метод анализа сетевого графика[x)] основан на принципе полного перебора. Обозначим через E_α множество вершин (событий) e_α сетевого графика, для которых $\Gamma^{-1} e_\alpha = \Theta$, и через E_ω множество вершин e_ω, для которых $\Gamma e_\omega = \Theta$. Таким образом, множество E_α есть множество начальных событий (входов), а множество E_ω – множество конечных событий (выходов графа).

Принцип полного перебора заключается в переборе всех путей, идущих от любого $e_\alpha \in E_\alpha$ к любому событию $e_\omega \in E_\omega$.

Для того чтобы начать полный перебор всех путей от e_α к e_ω, определяются множества E_α и E_ω. Положим вначале $\overline{T}_e = 0$ для всех событий и присвоим, кроме того, всем событиям $e_\alpha \in E_\alpha$ признак времени. Полный перебор начинается с произвольного события $e \in E_\omega$. Он состоит в выполнении конечного числа шагов влево с целью отыскания события, отмеченного признаком времени, и конечного числа шагов вправо с целью вычисления T_e для всех событий, не имеющих признака времени.

Опишем кратко эти шаги. Шаг влево – это переход от события e_1 к событию $e_2 \in \Gamma^{-1} e_1$ и проверка наличия признака времени у события e_2. Обозначим переход символом $(e_2 \leftarrow e_1)$ и назовем работу (дугу) (e_2, e_1) пройденной, если указанный шаг сделан.

Заметим, что за конечное число шагов влево

$$e_2 \longleftarrow e_1$$
$$e_3 \longleftarrow e_2$$
$$\cdots\cdots\cdots$$
$$e_{s+1} \longleftarrow e_s$$

будет найдена такая последовательность пройденных работ

$$(e_2, e_1), (e_3, e_2), \ldots\ldots, (e_{s+1}, e_s)$$

и, следовательно, будет рассмотрена последовательность событий

$$L = \{e_1, e_2, \ldots e_p, \ldots e_s, e_{s+1}\},$$

---

x) Изложение этого метода было сделано в устном сообщении Г.М.Адельсона-Вельского и Ф.М.Филлер на семинаре по математическим методам.

что либо

I) существует такое $e_\rho \in L$, что $e_\rho = e_{s+1}$,
либо 2) e_{s+1} имеет признак времени.

В первом случае нахождение T_e теряет смысл, так как сетевой график содержит контур $\mu = [e_{s+1}, e_s, \dots, e_\rho]$ (рис. I)

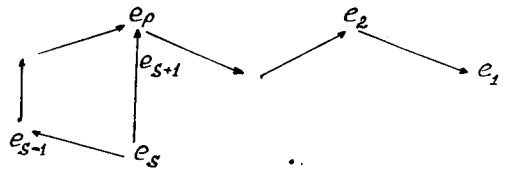

Рис. I.

и алгоритм работает для выявления всех контуров. Для этого произвольная дуга (работа), принадлежащая контуру, рвется, и к полученному новому сетевому графику применяется алгоритм полного перебора.

Во втором случае, когда найдено событие e_{s+1}, имеющее признак времени, совершаем шаг вправо. Шаг вправо (обозначим его символом $e_{s+1} \longrightarrow e_s$) заключается в определении величины

$$\widetilde{T}_s = max\left[\widetilde{T}_{s+1} + \tau_{s+1,\,s};\ \overline{T}_s\right]$$

(в начальный момент все $\overline{T}_s = 0$) и в установлении существования непройденных работ (e_κ, e_s). Если таковых нет, то, положив

$$T_s = \widetilde{T}_s,$$

присваиваем событию e_s признак времени. Если есть хотя одна непройденная ещё работа $(e_\kappa, e_s) \in U$, то признак времени событию e_s не присваивается и совершается шаг влево $e_\kappa \leftarrow e_s$.

Очевидно, что, отправляясь последовательно от каждого события, принадлежащего E_ω, мы, совершая указанные шаги, придем к тому, что все события получат признак времени, и соответствующие величины T_e будут найдены для всех событий e.

Для нахождения $\overline{T}_e$ данная последовательность шагов выполняется в обратном порядке. Для этого достаточно, например, совершить замену (i,j) на (j,i) для всех работ $(i,j) \in U$ и повторить указанные шаги.

Эффективное использование этого алгоритма на ЭВМ предполагает некоторую предварительную работу по перекодировке событий, сортировке множества работ U и нахождению множеств E_α и E_ω. Легко заметить, что алгоритм полного перебора может быть легко приспособлен для выдачи информации о структуре сетевого графика, необходимой для его вычерчивания. Чтобы пояснить это, рассмотрим разбиение множества событий E на классы.

Отнесем к нулевому классу те события e_o, для которых $\Gamma^{-1} e_o = \theta$. Таким образом, определенное выше множество E_α есть множество событий, принадлежащих нулевому классу E_o. К классу E_1 отнесем те события, для которых $\Gamma e \in E_o$.

Вообще событие e принадлежит нулевому классу событий E_κ, если существует работа $(i,e) \in U$, где i принадлежит первому классу $E_{\kappa-1}$.

Так, например, в сетевом графике, изображенном на рис. 2, к нулевому классу относятся события A и E, к первому классу - события F и C, ко второму классу - B и K, к третьему - H и G и т.д.

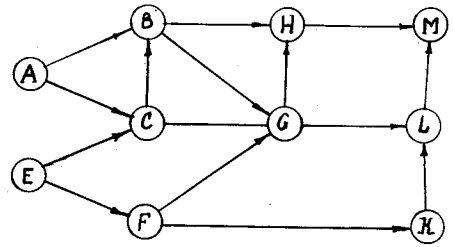

Рис. 2.

Положим теперь $\tau_{ij} = 1$ для всех работ $(i,j) \in U$. Легко видеть, что величины T_e, вычисленные по формуле $(*)$ примут следующие значения:

$$T_i = 0 \text{ для } i \in E_o,$$
$$T_i = 1 \text{ для } i \in E_1,$$
$$T_i = 2 \text{ для } i \in E_2,$$
$$\dots\dots\dots\dots$$
$$T_i = k \text{ для } i \in E_k.$$

Таким образом, любой алгоритм, позволяющий вычислить наиболее ранний срок начала событий T_e, дает возможность раз-

бить множество событий сетевого графика на классы. По-видимому, такая информация является достаточной для вычерчивания сетевого графика.

Алгоритм, основанный на принципе полного перебора, легко реализуется для сетевых графиков, информация о которых не требует привлечения внешней памяти ЭВМ. Для больших сетей каждый шаг влево требует вызова в оперативную память машины информации об отдельной работе, что связано с колоссальным числом обращений к внешней памяти.

Эту трудность можно преодолеть, если разбить сетевой график на подграфики, у которых число входов $|E_\alpha|$ и выходов $|E_\omega|$ невелико. Если такое разбиение возможно, то сетевой график анализируется по частям следующим образом. Рассмотрим выделенный подграф и при помощи алгоритма полного перебора найдем пути максимальной длины, ведущие от каждого начального события к каждому конечному.

Пусть одним из таких путей будет

$$\mu = [e_1, e_2, \ldots, e_\rho],$$

причем продолжительность этой последовательности работ будет T_ρ . Заменим каждый такой путь фиктивной работой (e_1, e_n) и положим для нее $\tau_{1\rho} = T_\rho$.

Так, например, на подграфе рис. 3

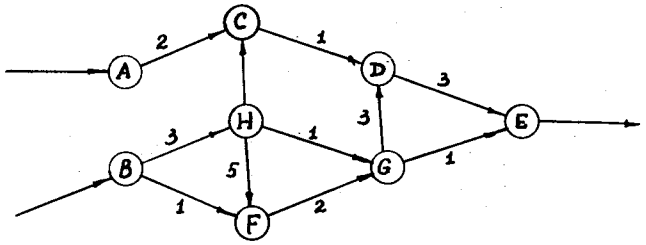

Рис. 3.

такими путями будут

$$\mu_1 = [A, C, D, E],$$

$$\mu_2 = [B, H, F, G, D, E],$$

а соответствующие им длины будут

$$\tau_1 = 2 + 1 + 3 = 6,$$

$$\tau_2 = 3 + 5 + 2 + 3 + 3 = 16.$$

Следовательно, фиктивными работами, вводимыми вместо этих путей, будут

(A,E) - с продолжительностью $\tau_{AE} = 6$,

(B,E) - с продолжительностью $\tau_{BE} = 16$.

Число фиктивных работ, очевидно, не может превзойти величину $|E_\alpha| \cdot |E_\omega|$, поэтому, если число работ (дуг) в данном подграфе больше произведения $|E_\alpha| \cdot |E_\omega|$, то замена этого подграфа фиктивными работами сокращает количество информации, необходимой для обработки всего сетевого графика.

Нетрудно заметить, что разбиение сетевого графика на подграфы с малым числом входов и выходов не всегда возможно (хотя и можно указать целый ряд разработок, при которых сетевые графики легко разбиваются на такие подграфы). Некоторые трудности разбиения сетевого графика возникают и от того, что оно должно, по-видимому, осуществляться "вручную", без использования ЭВМ, а это затруднительно, так как сетевой график в достаточно обозримой форме может и не существовать. Между тем для сетевых графиков, информация о которых может быть помещена в оперативной памяти ЭВМ, данный алгоритм достаточно эффективен, так как с его помощью можно выявить целый ряд ошибок, таких, как например, изолированные звенья, работы, лежащие на контурах и т.д.

Рассмотрим алгоритм полного перебора на следующем примере (рис. 4)

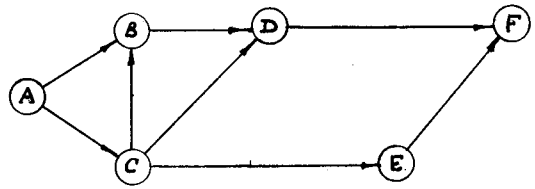

рис.4.

Пусть информация о сетевом графике, изображенная на рис.4, задана в виде, представленном на табл. I и 2.

Таблица I

I часть				2 часть		
N	i	j	τ_{ij}	a_i	b_j	τ_{ij}
I	B	D	I	k + 2	k + 4	I
2	C	D	2	k + 3	k + 4	2
3	D	F	6	k + 4	k + 6	6
4	E	F	3	k + 5	k + 6	3
5	A	B	I	k + I	k + 2	I
6	C	B	5	k + 3	k + 2	5
7	C	E	4	k + 3	k + 5	4
8	A	C	2	k + I	k + 3	2

Таблица 2

N	i	c_i	Признак времени	Признак конца	n_i	T
k + I	A	-	I	0	0	0
k + 2	B	5	0	0	2	0
k + 3	C	8	0	0	I	0
k + 4	D	I	0	0	2	0
k + 5	E	7	0	0	I	0
k + 6	F	3	0	I	2	0

Таблица I (таблица работ) состоит из двух частей. В первой части каждая работа задана тройкой чисел i, j, τ_{ij}, где i - код начала работы, j - код конца работы в исходной нумерации (на рис. 4 эти коды заменены буквами). Во второй части таблицы та же тройка чисел записана после перенумерации, i и j имеют уже удобные для представления в машине номера. Обычно эта "машинная" нумерация является сплошной. Здесь за номер события A условно принят новый - "$k+1$", B - "$k+2$" и т.д.

Таблица 2 (таблица событий) состоит из 7 столбцов. Первые два столбца отведены под номера событий - исходные и "машинные", соответственно. В третьем столбце для каждого события указывается номер той строки в табл. I, в которой впервые

встречается работа, имеющая своим концом рассматриваемое событие. Четвертый столбец отведен для признака времени: в строке ставится единица, если для данного события есть признак времени, а нуль - в противном случае. В пятом столбце единицей отмечаются выходы сетевого графика, т.е. те точки, с которых начинается процесс полного перебора. В шестом столбце ставится число событий, непосредственно предшествующих данному событию, т.е. количество элементов $|\Gamma_j^{-1}|$ для каждого события j. Последний столбец, служащий для занесения вычисляемых в процессе полного перебора времен T_e, сначала заполняется нулями. В таблице 2 находим событие, имеющее признак конца. В нашем случае это событие F. Соответствующее значение $c_i (c_6 = 3)$ указывает местонахождения работы (x, F) в системе работ, по которой совершен шаг влево (в нашем случае это работа (D, F)). Все работы, имеющие вид (x, F), ещё не пройдены. Совершим шаг влево по первой работе вида (x, F), т.е. по работе (D,F), причем отметим, что эта работа стала пройденной введением дополнительной строчки, которую назовём счётчиком

$$C_{r_1}: 1, k+6, F.$$

Определим, имеет ли событие D признак времени. Для этого отыскиваем в табл. 2 событие D (местонахождение события D в табл. 2 указано под a_i). Так как D не имеет признака времени, то совершаем шаг влево от события D по первой непройденной работе вида (x, D) (местонахождение этих работ указывает Ci) и заполняем C_{r_2}

$$C_{r_2}: 1, k+4, D.$$

Работа (B, D) становится пройденной. Повторяя эти шаги, мы дойдем до события A, имеющего признак времени, причем будет заполнен следующий массив счетчиков:

$$C_{r_1}: 1, k+6, F$$
$$C_{r_2}: 1, k+4, D$$
$$C_{r_3}: 1, k+2, B$$

Совершаем шаг вправо, вычисляя $T_B = 0 + I = I$, и вписываем это значение в седьмой столбец табл. 2. Проверим, возможен ли шаг влево. Так как для события B $n_i = 2$, а значение $C_{r_3} = 1$, то из двух работ вида (x, B) работа (A, B) оказалась пройденной. Поэтому, не приписывая событию B признака вре-

мени, совершаем шаг влево, причем C_3 принимает значение

$$C_3 : 2 , к+2, B .$$

Так как событие C не имеет признака времени, то совершается ещё один шаг влево, после чего массив счетчиков примет вид :

$$C_1 : 1 , к+6, F$$
$$C_2 : 1 , к+4, D$$
$$C_3 : 2 , к+2, B$$
$$C_4 : 1 , к+3, C .$$

Событие A имеет признак времени, поэтому совершаем шаг вправо и вычисляем $T_C = 0 + 2 = 2$. Для события C $n_i = 1$, а потому шаг влево невозможен, и событие C получит признак времени.

Совершаем шаг вправо, стирая счетчик C_4, и вычисляем

$$T_B^{(1)} = 2 + 5 = 7 ,$$

Так как $T_B^{(1)} > T_B$, то в табл. 2 вносим значение $T_B^{(1)} = 7 = T_B$. Но для события B $n_i = 2$ и $C_3 = 2$, поэтому событие получает признак времени и C_3 стирается.

Теперь уже всякий путь, начинающийся в вершине F, может закончиться в вершинах A, B, C (и только в них).

Продолжая далее уже описанную процедуру, мы придем к такому положению, когда все вершины графика получат признак времени и тем самым все искомые времена T_e будут найдены.

Существует очень простой метод анализа сетевого графика, который изложен в работе [1].

В основных чертах он состоит в вычислении величин T_e и T_e' по итерационным формулам:

$$T_e = \max_{i \in \Gamma_e^{-1}} \left[T_i + \tau_{ie} ; T_e \right] ,$$

$$T_e' = \max_{j \in \Gamma_e} \left[T_j' + \tau_{ej} ; T_e' \right]$$

при помощи просмотра массива работ. Легко заметить, что число итераций, а следовательно, и число просмотров массива работ не превосходят числа дуг N в пути, имеющем наибольшее число дуг в данном сетевом графике. Непосредственное применение этого метода дает возможность разбить множество вершин данного графика на классы, включающие в себя вершины одного и того же порядка.

Если положить $\tau_{ij} = 1$ для всех работ $(i,j) \in U$ и вести счет по неупорядоченному списку работ (формула (I)), то очевидно, все события e , принадлежащие i -му классу, будут иметь $T_e' = i$.

Этот метод не заключает в себе возможности автоматического отыскания контуров. Однако если анализируемый граф содержит контуры, то указанный итерационный процесс расходится. Следовательно, из того, что итерации идут слишком долго, можно усмотреть наличие контуров.

Число итераций может быть сокращено при специальном упорядоченном просмотре множеств работ U или, что то же самое, при наличии упорядоченного списка работ.

Будем говорить, что список работ $(i,j) \in U$ логически упорядочен, если работа (i,j) встречается в этом списке после всех работ вида (k,i) , т.е.

$$(k,i) \prec (i,j) \text{ для всех } i \in E . \qquad (3)$$

Если список работ логически упорядочен, то, очевидно, уже после одной итерации можно будет определить все числа T_e , но для получения чисел T_e' необходимо сделать N итераций. Чтобы получить числа T_e' также за одну итерацию, достаточно совершить просмотр логически упорядоченного списка работ в обратном порядке, так как работа (i,j) , что следует из (3), в логически упорядоченном списке встречается только тогда, когда все работы вида (j,k) стоят в этом списке ниже, т.е.
$(i,j) \prec (j,k)$ для всех $j \in E$.

Существует несколько способов получения логически упорядоченного списка работ. Рассмотрим некоторые из них, которые, на наш взгляд, легко реализуются на ЭВМ.

Пусть даны сетевой график (рис. 5)

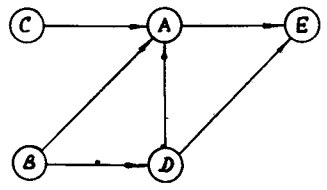

Рис. 5.

и список работ и событий, причем список работ упорядочен по коду начала работы. Пусть, кроме того, для каждой работы (i,j) известен порядковый номер события j в списке событий (столбец в табл. 3). Отметим некоторым признаком, например I, конец каждой группы работ, имеющих один и тот же код, начало работ (получение такого списка работ назовем "сортировкой работ по началу"). В списке событий отметим для каждого события число неупорядоченных предшественников (колонка e табл. 4) и порядковый номер событий в списке работ (i,j), где это событие встречается впервые.

Алгоритм перенумерации заключается в том, что каждому событию присваивается некоторый новый номер S (т.е. создается "словарь"), удовлетворяющий следующему требованию: если событие i предшествует событию j, $i \prec j$, то новый номер события, равный S_i, должен удовлетворять условию:

$$S_i < S_j,$$

где S_j - новый номер события j.

Алгоритм работает таким образом, что перенумерация событий происходит тогда и только тогда, когда все предшествующие события перенумерованы. Для этой цели как раз и необходима колонка c (табл. 4). Если $c_i = 0$, где i - номер события в списке, то это значит, что все события, предшествующие данному, уже перенумерованы и далее можно осуществлять перенумерацию данного события.

Таблица 3

N	(i,j)	a	b
I	A E	I	$\kappa + 5$
2	B A	0	$\kappa + I$
3	B D	I	$\kappa + 4$
4	C A	I	$\kappa + I$
5	D A	0	$\kappa + I$
6	D E	I	$\kappa + 5$

Таблица 4

N	i	c	d
$\kappa + I$	A	3	I
$\kappa + 2$	B	0	2
$\kappa + 3$	C	0	4
$\kappa + 4$	D	I	5
$\kappa + 5$	E	2	-

Таким образом, для осуществления перенумерации событий просматривается список событий и находится первое событие, для которого $c_i = 0$. Событие i может быть занумеровано очередным натуральным числом (в данном случае при кодовом просмотре событие B получает номер I). Для того чтобы продолжить процесс перенумерации событий, для которых событие i (в данном случае B) было предшествующим, необходимо изменить число предшествующих неперенумерованных событий c_i. Для этого в списке работ по числу d_i (в данном случае $d_i = 2$) находим начало группы работ вида (i,j), причем b_j (в данном случае $b_j = \kappa + 1$) указывает номер события i, для которого число c_{i_1} необходимо уменьшить на единицу. Событие i_1 непосредственно предшествует всем тем событиям j, для которых существуют работы вида (i,j). Если $a_j = 0$, то это значит, что не вся группа работ (i,j) уже просмотрена и, следовательно, для нахождения следующего $j = i_2$ необходимо спуститься по списку работ ниже, вычитая при этом из $c_{i_2}, \ldots$ по единице (в нашем случае $a_2 = 0$, но $a_3 = 1$, и значит, $(c_1' = c_1 - 1 = 2,\ c_4' = c_4 - 1 = 0)$.

Легко заметить, что при работе данного алгоритма коды событий $i \in E$ и коды работ (i,j) не участвуют, поэтому для реализации алгоритма достаточно работать с более компактными списками работ и событий.

Таблица 5

N	a	b
I	I	$\kappa + 5$
2	0	$\kappa + I$
3	I	$\kappa + 4$
4	I	$\kappa + I$
5	0	$\kappa + I$
6	I	$\kappa + 5$

Таблица 6

	c	d
$\kappa + I$	3	I
$\kappa + 2$	0	2
$\kappa + 3$	0	4
$\kappa + 4$	I	5
$\kappa + 5$	2	-

Во время первого просмотра списка событий получаем такую последовательность значений c_i и d_i:

Таблица 7

	c	d_{ns}
$\kappa + I$	2	I
$\kappa + 2$	x	I^x
$\kappa + 3$	0	4
$\kappa + 4$	0	5
$\kappa + 5$	2	-

Таблица 8

	c	d_{ns}
$\kappa + I$	I	I
$\kappa + 2$	x	I^x
$\kappa + 3$	x	2^x
$\kappa + 4$	0	5
$\kappa + 5$	2	-

В зависимости от конкретных данных ЭВМ и размеров сетевого графика запись информации можно видоизменить. Так,например, можно объединить оба списка в следующий вид:

Таблица 9

	(i,j)	i	c	ℓ	признак ns
I	A E	A	3	7	
2	B A	B	0	I	
3	B D	–	–	5	
4	C A	C	0	I	
5	D A	D	I	I	
6	D E	–	–	7	
7		E			

Здесь под признаком понимается некоторый символ, указывающий, что данное событие перенумеровано, а признаком $a_i = 0$ служит $c_i \neq 0$.

Алгоритм легко видоизменить так,чтобы одновременно с перенумерацией событий производилась группировка (или нумерация событий) по классам, которая является наиболее удобным видом информации для ручного построения графика.

На той же идее основано построение несколько упрощенного алгоритма, описанного в работе [2]. Здесь вместо числа неупорядоченных предшественников каждому событию ставится признак запрета, который снимается лишь тогда, когда данное событие уже перенумеровано. Событие можно перенумеровать только тогда, когда со всех непосредственно предшествующих ему событий признак запрета снят. Исходная таблица на примере того же сетевого графика примет вид (табл. I0 и II)

Таблица I0

N	(i,j)	i	j
I	C A	$\kappa+3$	$\kappa+1$
2	B A	$\kappa+2$	$\kappa+1$
3	D A	$\kappa+4$	$\kappa+1$
4	A E	$\kappa+1$	$\kappa+5$
5	D E	$\kappa+4$	$\kappa+5$
6	B D	$\kappa+2$	$\kappa+4$

Таблица II

	i	Запрет	S
$\kappa+1$	A	I	–
$\kappa+2$	B	0	I
$\kappa+3$	C	0	2
$\kappa+4$	D	I	–
$\kappa+5$	E	I	–

Здесь список работ рассортирован по кодам конца работ, а в списке событий единицей отмечен признак запрета.

Просматривая список работ (i,j), находим те события j, для предшественников которых признак запрета снят, присваиваем ему номер, равный очередному натуральному числу, и снимаем с события j признак запрета. Легко видеть, что в данном случае нумерация событий не будет соответствовать разбиению множества событий на классы. Просмотры списка работ продолжаются до тех пор, пока все события не будут перенумерованы.После перенумерации событий производится сортировка списка работ по коду начального (конечного) события.

Под последней понимается получение такого списка, в котором работы располагаются в порядке возрастания величины кода начального (конечного) события.

Предлагаемый алгоритм сортировки состоит в последовательном просмотре каждого двоичного разряда кода начального (конечного) события (табл. I2). Просмотр начинается со старшего двоичного разряда каждого кода, в результате чего работа с рассматриваемым кодом попадает в верхний "ящик" (верхнюю часть списка), если этот двоичный разряд содержит нуль, и в нижний "ящик" – в противном случае. Конец каждого "ящика" отмечается признаком. Переход от одного разряда к другому разбивает на два каждый полученный на предыдущем шаге "ящик" по указанному выше признаку. Заметим, что при таком разбиении один из "ящиков" может оказаться пустым.

Таблица I2

	(i,j) в 8-ом коде	i в двоичном коде	I	2	...
I	03 – 04	0011 – 04	0011 – 04	0011–04	...
2	I3 – 0I	I011 – 0I	0I0I – 06	0011–05	...
3	06 – I0	0II0 – I0	0II0 – I0	0II0–I0	...
4	03 – 05	0011 – 05	0011 – 05	0I0I–06	...
5	05 – 06	0I0I – 06	I011 – 0I	I011–0I	...
6	I7 – I2	IIII – I2	IIII – I2	IIII–I2	...

Для получения числа c_i неупорядоченных предшествующих событий для каждого события i достаточно выполнить сорти-

ровку списка работ по конечному событию, после чего c_i легко может быть сосчитано.

Нетрудно видеть, что полученный список будет логически упорядоченным.

Предложенный метод сортировки и упорядочения удобен для целого ряда часто встречающихся на практике случаев. Руководители разработок довольно больших систем часто высказывают пожелание, чтобы под код события было отведено достаточно большое десятичное число, с тем, чтобы диапазон кодов событий был примерно в 100 раз больше, чем фактическое число событий. Так, например, если сетевой график содержит 5000 событий, то коды событий обычно кодируются 6-7-значными числами.Но,с другой стороны, такой код неудобен для реализации сетевого графика на ЭВМ, так как информация об одной работе не может быть помещена в одной ячейке памяти. Кроме того, при анализе сетевого графика удобно,чтобы коды события, являющиеся кодом начала и конца работы, указывали номера тех ячеек памяти, где хранится информация о событии, как например, T_e и T_e'.

Успешная разработка сложных систем требует, чтобы общая последовательность работ, входящих в систему, оставалась некоторое время стабильной. Это значит, что логические структуры соответствующих данной системе сетевых графиков в течение определенного времени тождественны и отличаются лишь временными оценками. Для всех этих сетевых графиков логически упорядоченный список работ один,и тот же и поэтому информацию об этом списке имеет смысл хранить во внешней памяти (на ленте или на перфокартах). Этой цели может служить составленный при логическом упорядочении "словарь", дающий возможность привести в упорядоченный вид входную информацию о сетевом графике.

Если сеть сложна, вероятность допущения ошибки при наброске сетевого графика становится значительной. Наиболее часто ошибки возникают при первоначальном наброске сетевого графика и при изменении его логической структуры, т.е. при выяснении новых работ, устранении лишних работ, раздроблении крупных работ и т.д. Во всех этих случаях речь идет о построении нового графика, а поэтому при анализе сетевого графика на ЭВМ требуется, чтобы была выдана такая информация, которая позволила бы легко вычертить этот граф на бумаге. Для этой цели наиболее удобно упорядочить события по классам.Типичные ошибки, возникающие при наброске сетевого графика,следующие: I) в сетевом графике появляются контуры, 2) соответствующий граф

имеет несколько компонент связности, 3) сетевой график имеет много лишних начальных и конечных событий.

Множества $E_\varkappa$ начальных и E_ω конечных событий могут быть определены заранее, поэтому они связаны с реально зафиксированным перечнем этих событий в начальный момент счета, а именно: при сортировке списка работ по коду начального и конечного события для составления "словаря". Если полученные при сортировке множества $E_\varkappa$ и E_ω не совпадают с исходными, то следует предусмотреть останов ЭВМ. Так как кроме этого требуется ещё выдать информацию о классах событий, то следующим этапом работы ЭВМ будет счет T_e и T_e' по неупорядоченному списку, с фиктивными длительностями работ $\tau_{ij}^{\circ} = 1$ для всех работ (i,j), принадлежащих U. Программа в этом случае должна быть построена так, чтобы число итераций не превышало заранее заданного числа. Это число можно оценить ориентировочно, исходя из следующих соображений. Разработка системы рассчитана на определенное число лет. Если продолжительность каждой работы принять равной одной неделе, то в данном сетевом графике легко можно указать максимальное число N классов событий. Например, число классов сетевого графика, соответствующего разработке системы на 2-3 года, не может превзойти числа 200. Так как для всех событий $e \in E_\kappa$ при $\tau_{ij} = 1$ для всех $(i,j) \in U$, $T_e = \kappa$, то появления хотя бы одного $T_e > N$ указывает, что сетевой график содержит контуры. Работы, образующие контур, можно выявить, например, следующим образом.Если программа построена по схеме одновременного счета T_e и T_e' по неупорядоченному списку работ, то можно выделить те события, для которых $T_e > N$ и $T_e' > N'$. Очевидно, что $T_e > N$ как для подмножества событий $\{j\} \in E'$, которые принадлежат контуру, так и для подмножества событий $\{j''\} \in E$, для которых $j' \prec j''$. Аналогично, $T_e' > N$ для подмножества событий $\{i'\} \in E$, принадлежащих контуру, и всех тех событий $\{i''\} \in E$, для которых $i'' \prec i'$. Пусть $A = \{j'\} \cup \{j''\}$ и $B = \{i'\} \cup \{i''\}$, так что $A \subset E$ есть подмножество событий, для которых $T_e > N$ и $B \subset E$ - подмножество событий, для которых $T_e' > N$. Тогда, очевидно, событие, принадлежащее контуру, принадлежит и пересечению этих подмножеств $A \cap B$. Но одно только указание множеств без привлечения ЭВМ не дало бы возможности эффективно найти содержащиеся в сетевом графике контуры, что видно из рис. 6 следующего сетевого графика.

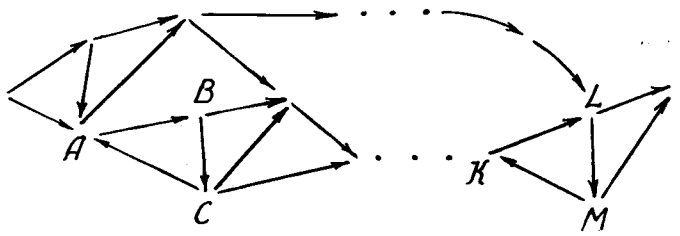

Рис. 6.

Здесь множество $A \cap B$ включает в себя почти все события сетевого графика, хотя только 6 событий принадлежат контурам A, B, C и K, L, M.

Предлагаемый нами способ выделения контуров сетевого графика в основных чертах можно представить себе как некоторое добавление к описанному выше алгоритму по разбиению событий на классы.

Положив все $\tau_{ij} = 1$ и найдя величины T_e и T_e' по итеративным формулам:

$$T_e = \max\left[T_e, T_i + 1\right], \quad i \in \Gamma_e^{-1},$$

$$T_e' = \max\left[T_e', T_j + 1\right], \quad j \in \Gamma_e,$$

мы можем для каждого события на каждой итерации запомнить номера $i(e)$ и $j(e)$, на которых достигаются указанные максимумы.

Как было указано выше, при наличии контура в графе число итераций при вычислении величин T_e и T_e' становится сколь угодно большим. Далее, если некоторая вершина e входит в контур, то вершины $i(e)$ и $j(e)$ также входят в контур. Исходя из этого, можно в некоторый момент процесса, когда число итераций превзойдет заданное число N, выбрать из списка событий любое событие e, для которого $T_e > N$, и построить, начиная с этого события, последовательность вершин: $(e, e_1, e_2, \ldots, e_{\kappa+1} \ldots)$, где каждая $e_{\kappa+1} = i(e_\kappa)$.

В процессе построения этой последовательности номер каждой вводимой вершины $e_{\kappa+1}$ сравнивается с номерами каждой из предшествующих вершин в этой последовательности. Если при этом

окажется, что $e_{\kappa+1} = e_s$ $(s \le \kappa)$, то последовательность вершин $(e_s, e_{s+1}, \ldots, e_{\kappa+1})$ образует контур.

Литература

I. Петрова Л.Т., Карнаухова Н.Н. Об одном алгоритме нахождения критического пути сетевого графика.(Данный сборник).

2. Лейфман Л.Я., Петрова Л.Т. Некоторые алгоритмы для анализа ориентированных графов.(Данный сборник).