УДК 519.8

В.М. Котов[1], Х. Келлерер[2], Н. Браунер[3], Г. Финке[3]

АЛГОРИТМ ДЛЯ ВЕРСИЙ ЗАДАЧИ $Pm||C_{\max}$ С НЕПОЛНОЙ ИНФОРМАЦИЕЙ

Исследуются две модели с неполной информацией для задач теории расписаний с идентичными процессорами. Предлагается общая параметрическая схема построения решений для таких задач. Достигаются рекордные гарантированные оценки точности для алгоритмов при соответствующих параметрах, которые доказывают принципиальное различие моделей.

Введение

Построение расписаний на m идентичных процессорах, когда требуется распределить работы таким образом, чтобы время завершения последней работы было минимально, является классической задачей теории расписаний [1]. Хорошо известно, что эта задача является NP-трудной. Для ее решения предложено множество приближенных алгоритмов.

В последнее время большой интерес вызывают версии задачи с неполной информацией, так называемые онлайн и semi онлайн. В онлайн-версии нет никакой информации о количестве работ, длительности их выполнения на процессоре. Работы поступают одна за другой, и, прежде чем поступит информация о длительности следующей работы или об ее отсутствии, текущая работа должна быть назначена на конкретный процессор, причем это назначение окончательное, т. е. не может быть изменено в дальнейшем. Для оценки качества работы алгоритмов решения версий задач с неполной информацией обычно используется сравнительный анализ. Сравниваются значения решений, получаемых алгоритмом для онлайн-версии, с оптимальными значениями соответствующей офлайн-версии (когда все данные заранее известны), и устанавливается гарантированная оценка алгоритма [2, 3]. Следует отметить, что при анализе алгоритмов для задач с неполной информацией используются так называемые нижние границы для гарантированных оценок онлайн- и semi онлайн-алгоритмов [4, 5]. Наличие нижней границы означает принципиальную невозможность построения алгоритма с гарантированной оценкой, лучшей, чем нижняя граница.

Semi онлайн-версия задач предполагает наличие дополнительной информации общего характера [6]. Для рассматриваемой задачи известно несколько semi онлайн-версий: работы поступают в отсортированном порядке, известны оптимальное значение решения (задача 1) [7] и общая сумма длительностей выполнения (задача 2) [8].

Общая схема для задач с известным оптимальным значением целевой функции или суммарной длительностью всех работ

Будем считать, что Z – оптимальное значение целевого функционала, а S – общая сумма длительностей выполнения работ для semi онлайн-версий задачи. Тогда нижняя граница LB для оптимального значения функционала в первом случае равна Z, а во втором – средней загрузке процессоров S/m.

Будем считать, что $LB = 1$. Этого можно добиться, пронормировав все величины (поделив на значение LB). Определим параметр $\alpha > 0$ и, используя этот параметр, опишем алгоритм, гарантированная оценка которого будет $(1+\alpha)$. В работе [8] было доказано, что нижняя граница для semi онлайн-версии, когда известна общая сумма процессорных времен, не меньше 1,5, поэтому $\alpha \geq 0{,}5$.

Разобьем поступающие работы на классы в соответствии с их процессорным временем.

В класс A определим работы, у которых процессорное время больше чем $(1+\alpha)/2$. Очевидно, что для обеспечения нужной гарантированной оценки нельзя назначать две работы из класса A на один процессор.

В класс *B* определим работы, у которых процессорное время больше α, но не больше чем (1+α)/2. Очевидно, что для обеспечения нужной гарантированной оценки на один процессор можно назначать две работы из класса *B*. При этом загрузка такого процессора будет больше 1.

В класс *C* определим работы, у которых процессорное время не больше чем α. Добавление таких работ на процессор с загрузкой меньше *LB* приведет к тому, что загрузка процессора может превысить 1, но не превысит (1 + α).

Понятно, что в любом решении невозможна ситуация, когда загрузка каждого процессора превосходит 1. Основная идея алгоритма состоит в специальным образом осуществляемой загрузке процессоров, чтобы для одного процессора или группы процессоров их средняя загрузка была не меньше 1, а максимальная загрузка любого из процессоров не превосходила более чем в (1 + α) раз величину оптимального значения целевой функции.

Ключевым моментом данной схемы является выделение специальных типов процессоров:

– процессор относится к типу *Close_A*, если на него назначена одна работа из класса *A* и несколько работ из класса *C*, причем общая загрузка процессора больше 1, но не превышает величины (1 + α);

– процессор относится к типу *Open_A*, если на него назначена одна работа из класса *A* и несколько работ из класса *C*, причем общая загрузка процессора не больше 1;

– процессор относится к типу *Close_B*, если на него назначены две работы из класса *B*;

– процессор относится к типу *Open_B*, если на него назначена одна работа из класса *B*.

Для работ из класса *C* определим группу из четырех процессоров (пусть это процессоры *CC*1, *CC*2, *CC*3, *CC*4), которую будем называть *Close_bunch_C*. Такую группу можно получить следующим образом. Пусть поступают работы из класса *C*. При поступлении очередной работы из класса *C* назначаем ее на процессор *CC*1 группы, если получаемая суммарная загрузка не превышает α. Если же суммарная загрузка процессора превысит α, то пытаемся назначить работу на процессор *CC2*, чтобы суммарная загрузка не превысила α. Если это возможно, то назначаем. В случае когда невозможно назначить работу ни на процессор *CC*1, ни на *CC*2 без превышения α, назначаем ее на процессор *CC*3, добавляем к группе пустой процессор *CC*4 и считаем, что группа сформирована.

Очевидно, что в сформированной группе суммарная загрузка на первых двух процессорах больше α. Это справедливо также для первого и третьего, а также для второго и третьего процессоров.

Свойство. Загрузка каждого из процессоров *CC*1, *CC*2, *CC*3 группы не превосходит α, процессор *CC*4 пуст, а суммарная загрузка процессоров *CC*1, *CC*2, *CC*3 больше 3α/2.

Может оказаться, что будет сформирована неполная группа четвертого типа, когда в ней менее четырех процессоров. Такую группу будем называть *Open_bunch_C*. Группа этого типа может содержать от одного до трех процессоров, причем три процессора в ней могут быть только в том случае, если не осталось ни одного пустого процессора для формирования группы *Close_bunch_C*.

Алгоритм построения расписания состоит из двух этапов. На первом этапе формируются процессоры описанных выше типов следующим образом.

Если поступает работа из класса *A*, то приоритетным процессором для нее является наиболее загруженный процессор из группы *Open_bunch_C*. В этом случае получается процессор либо типа *Close_A*, либо *Open_A*. Если группы *Open_bunch_C* нет, но есть группа *Close_bunch_C*, то приоритетным для нее является наиболее загруженный процессор из данной группы. В этом случае формируется процессор типа *Close_A* и группа *Open_bunch_C*, которая получается путем удаления из нее пустого процессора. Иначе работа назначается на пустой процессор. Следует отметить, что процессорное время работы может быть больше 1. Назначение такой работы на любой из перечисленных выше процессоров (в том числе и на пустой) приводит к получению процессора *Close_A*, причем его суммарная загрузка не превосходит (1+α)*LB*.

Если поступает работа из класса *B*, то приоритетным для нее является процессор *Open_B*. Иначе работа назначается на пустой процессор. В этом случае получается процессор либо типа *Close_B*, либо *Open_B*.

Если поступает работа из класса *C*, то приоритетным для нее является процессор *Open_A*. В этом случае получается процессор либо типа *Close_A*, либо *Open_A*. Если такого процессора

нет, то работа назначается на процессор из незавершенной группы *Open_bunch_C* с целью получения группы *Close_bunch_C*. Иначе работа назначается на пустой процессор и открывается новая группа *Open_bunch_C*.

Первый этап алгоритма заканчивается, когда для очередной работы нет пустого процессора (не считая пустых процессоров в группах типа *Close_bunch_C*, которые на первом этапе не заполняются).

Очевидно, что после первого этапа могут быть одновременно получены процессоры *Close_A*, *Open_A*, *Close_B*, *Open_B* и группы процессоров *Close_bunch_C* и *Open_bunch_C*. При этом количество процессоров *Open_B* не превышает 1, а если есть группы процессоров *Close_bunch_C* или *Open_bunch_C*, то количество процессоров *Open_A* не превышает 1.

В дальнейшем процессоры *Close_A* и *Close_B* редуцируются, так как их загрузка больше 1. Отметим, что при назначении поступившей работы всегда имеется процессор, загрузка которого меньше 1.

Рассмотрим два возможных случая структуры распределения процессоров по типам после завершения первого этапа алгоритма.

Случай 1. Имеется более одного процессора типа *Open_A*. Это означает, что групп процессоров типа *Close_bunch_C* и *Open_bunch_C* нет. Действительно, пусть имеются два процессора типа *Open_A*, *AA*1 и *AA*2. Группа процессоров типа *Close_bunch_C* или *Open_bunch_C* не может быть построена после получения *AA*1 и *AA*2, так как работы из класса *C* были бы назначены на эти процессоры. С другой стороны, только один процессор из группы *Open_bunch_C* может иметь загрузку меньше $\alpha/2$.

Таким образом, процессоры могут быть только типа *Open_A* и не более одного процессора *Open_B*, остальные процессоры будут *Close_A* и *Close_B*.

Если в дальнейшем поступают работы из класса *C*, то они назначаются на наиболее загруженный процессор типа *Open_A* для получения процессора *Close_A*.

Если поступила работа длительности *X* из класса *B*, то пытаемся назначить ее на процессор типа *Open_B*, если он есть. Предположим, что процессора типа *Open_B* нет. Это означает, что можно пересчитать нижнюю оценку оптимального решения.

Действительно, имеется, по крайней мере, $m+1$ работа из классов *A* или *B*, причем одна работа из класса *B* должна быть назначена на процессор вместе с работой из класса *A* либо на один процессор должны быть назначены две работы из класса *A* или три работы из класса *B*. Следовательно, $LB \geq \min\{3\alpha, \alpha+(1+\alpha)/2, 1+\alpha\} = \alpha+(1+\alpha)/2$ при $1/2 \leq \alpha < 1$.

Заметим, что при назначении работы длительности *X* на наименее загруженный процессор его загрузка не превысит $1+(1+\alpha)/2$. Отсюда

$$(1+(1+\alpha)/2)/LB \leq (1+(1+\alpha)/2)/(\alpha+(1+\alpha)/2) = 1+(1-\alpha)/(\alpha+(1+\alpha)/2) \leq 1{,}5.$$

Если поступила работа длительности *X* из класса *A*, то можно пересчитать *LB*, при этом ее процессорное время может быть больше 1. Для этого определим новое значение $LB = \max\{\alpha+(1+\alpha)/2, X\}$.

Тогда при назначении работы длительности *X* на наименее загруженный процессор его загрузка не превысит $1+X$. Из этого следует

$$(1+X)/LB \leq 1+1/\max\{\alpha+(1+\alpha)/2, X\}.$$

Поэтому $(1+X)/LB \leq 1+\alpha$ для $\alpha \geq 2/3$.

Лемма. При отсутствии групп процессоров типа Close_bunch_C и Open_bunch_C при $\alpha \geq 2/3$ для любой последовательности работ назначение работы из классов *A* и *B* на наименее загруженный процессор обеспечивает загрузку, которая не превышает $(1 + \alpha)LB$.

Замечание. Для semi онлайн-версии, когда известно оптимальное значение целевой функции, в случае 1 после первого этапа будут поступать только работы из класса *C*, так как общее количество работ из классов *A* и *B* не превышает *m* (длительность каждой такой работы больше чем $Z/2$).

Случай 2. Имеется не более одного процессора типа *Open_A* и имеются группы процессоров типа *Close_bunch_C* и/или *Open_bunch_C* .

На втором этапе группы *Close_bunch_C* больше не меняются, а могут только редуцироваться. Группа *Close_bunch_C* будет считаться редуцированной, если сумма длительностей работ на ее процессорах не меньше четырех. Редуцированная группа из дальнейшего рассмотрения исключается, на процессоры такой группы новые работы не назначаются.

Идея второго этапа алгоритма основывается на использовании резервного четвертого процессора группы *Close_bunch_C*, что позволяет гарантировать суммарную загрузку в каждой группе процессоров типа *Close_bunch_C* не меньше четырех.

Отметим, что при поступлении работ из класса *A* они сначала назначаются на процессоры из группы *Open_bunch_C* .

Предположим, что имеются, по крайней мере, две нередуцированные группы процессоров типа *Close_bunch_C*. Группу *Close_bunch_CA* будем использовать для работ из класса *A*, а группу *Close_bunch_CB* – для работ из классов *B* и *C*.

Пусть в группу *Close_bunch_CA* были добавлены четыре работы из класса *A*. Тогда для редуцирования этой группы достаточно, чтобы суммарная загрузка в группе была не меньше четырех. Это значит, что с учетом свойства 1 должно выполняться соотношение $3\alpha/2+4(1+\alpha)/2\geq 4$. Отсюда следует, что редуцирование группы возможно при $\alpha\geq 4/7$.

Рассмотрим редуцирование группы *Close_bunch_CB*.

Пусть $L(CC)$ соответствует загрузке процессора *CC*. Не умаляя общности, будем считать, что загрузки процессоров *CC*1, *CC*2, *CC*3 упорядочены в порядке $L(CC1)\leq L(CC2)\leq L(CC3)$.

Вначале покажем, что при добавлении в группу более пяти работ класса *B* она может быть редуцирована. Если добавлено не менее шести работ класса *B*, то суммарная загрузка будет, как минимум, $3\alpha/2+6\alpha\geq 4$ при $\alpha\geq 1/2$. Поэтому будем рассматривать случай, когда поступило последовательно до пяти работ класса *B*.

Первую поступившую работу длительности $X0$ назначаем на процессор *CC*1. Если его загрузка не превысила α, то снова имеем группу *Close_bunch_C*. При этом опять будем считать, что загрузки процессоров *CC*1, *CC*2, *CC*3 упорядочены в порядке $L(CC1)\leq L(CC2)\leq L(CC3)$.

Пусть загрузка процессора *CC*1 стала больше α. При поступлении второй работы поступаем следующим образом.

Вариант 1. Если поступила работа длительности $X1$ из класса *B*, то пытаемся назначить ее на процессор CC1. Если суммарная загрузка не превысила $1+\alpha$, то делаем назначение. При этом загрузка процессора CC1 стала больше $\alpha/2+(1+\alpha)/2>1$ при $\alpha\geq 1/2$. При поступлении работ из классов *B* и *C* на пустой процессор может быть назначено не менее двух работ. Поэтому если эти работы не могли быть назначены на процессоры *CC*2 и *CC*3, то суммарная загрузка процессоров *CC*2, *CC*3 и *CC*4 будет больше $2(1+\alpha)$. Суммарная загрузка в группе будет больше четырех при $\alpha\geq 1/2$.

Вариант 2. Пусть поступила работа длительности $X1$ из класса *B*, но $L(CC1)+X1>1+\alpha$. В этом случае назначаем работу длительности $X1$ на процессор *CC*3. При этом очевидно, что работа длительности $X0$ была из класса *B*. Учитывая свойство 1, имеем $L(CC1)+L(CC3)+X1>1+\alpha+\alpha/2$. Поэтому суммарная загрузка одного из процессоров будут больше чем $(1+\alpha+\alpha/2)/2$. Не умаляя общности, пусть это процессор *CC*1. Тогда опять для трех процессоров *CC*2, *CC*3 и *CC*4 можно обеспечить суммарную загрузку больше $2(1+\alpha)$. Поэтому суммарная загрузка в группе будет больше $2(1+\alpha)+(1+\alpha+\alpha/2)/2$. Для редуцирования требуется выполнение неравенства $2(1+\alpha)+ (1+\alpha+\alpha/2)/2\geq 4$, которое справедливо при $\alpha\geq 6/11$. Отсюда следует, что редуцирование группы возможно при $\alpha\geq 6/11$.

Вариант 3. Поступила работа длительности $X1$ из класса *C*. Назначаем ее на прибор *CC*2. Будем считать, что $L(CC2)>\alpha$, иначе опять процессоры *CC*2 и *CC*3 можно пересортировать. После этого все поступающие работы из класса *C* будем назначать на процессор *CC*1, пока его загрузка не превысит 1. После этого имеем ситуацию, описанную выше. Пусть пришла работа длительности $X2$ из класса *B*. Пробуем назначить ее на процессор *CC*2. Если это возможно, то назначаем ее, получаем процессор с загрузкой больше 1. Поэтому получили вариант, описанный выше. Предположим, что $L(CC2)+X2>1+\alpha$. Учитывая $L(CC3)\geq L(CC2)$ до назначения работы длительности $X1$ на процессор *CC*2 и $X1\leq\alpha$, получаем $1< L(CC3)+X2\leq 1+\alpha$. Поэтому существует процессор с загрузкой больше 1. Аналогичный вариант описан выше.

Рассмотрим случай, когда осталась одна нередуцированная группа. При этом всегда можно получить процессор, загрузка которого больше 1. Действительно, при поступлении работы из класса *A* используется процессор *CC*3, для работ из классов *B* и *C* используются процессоры *CC*1 и *CC*4. Как только получен процессор с загрузкой, большей 1, остается не более трех процессоров. Если среди них есть два процессора с суммарной загрузкой больше 3/2, то оставшаяся максимальная возможная загрузка четвертого процессора не превышает 3/2, поэтому возможные оставшиеся работы могут быть гарантированно назначены на него. Если такой пары нет, то загрузка каждого из двух не превышает α. Если невозможно назначить работу на один из процессоров, то она может быть назначена на другой.

Таким образом, при любой последовательности поступления работ происходит редуцирование процессоров с суммарной загрузкой больше 1 или редуцирование группы со средней загрузкой больше 1, что гарантирует корректную работу алгоритма. При этом редуцирование групп процессоров возможно при $\alpha=\max\{4/7,6/11\}=4/7$.

Из этих рассуждений вытекают следующие утверждения:

Теорема 1. *При наличии непустой группы процессоров типа Close_bunch_C для любой последовательности длительностей работ при $\alpha=4/7$ существует алгоритм, обеспечивающий редуцирование.*

Теорема 2. *Предложенная двухэтапная схема корректно решает задачу с известной общей суммой длительностей выполнения работ при $\alpha\geq 2/3$, а задачу с известным оптимальным значением целевой функции при $\alpha\geq 4/7$.*

Заключение

В работе [9] для задачи с известной общей суммой длительностей выполнения недавно доказана нижняя граница, которая лежит в пределах интервала [1,58504; 1,58505]. Предложенный в настоящей работе алгоритм для задачи с известным оптимальным значением целевой функции имеет гарантированную оценку, которая лучше нижней границы для задачи с известной суммой длительностей выполнения. Это указывает на принципиальное различие задач.

В приведенной схеме не учитывается специфика задач. Например, для задачи с известным оптимальным значением целевой функции суммарное количество работ из классов *A* и *B* не превосходит *m*.

Дальнейшим возможным направлением исследований представляется разработка модифицированных алгоритмов, которые будут учитывать специфику каждой из задач, что, возможно, позволит существенно улучшить гарантированные оценки приведенной общей схемы.

Работа выполнена при частичной финансовой поддержке БРФФИ (проект № Ф10ФП-001).

Список литературы

1. Graham, R.L. Bounds for certain multi-processing anomalies / R.L. Graham // Bell System Technical J. – 1966. – Vol. 45. – P. 1563–1581.

2. Galambos, G. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling / G. Galambos, G. Woeginger // SIAM J. on Computing. – 1993. – Vol. 22. – P. 349–355.

3. Fleischer, R. Online scheduling revisited / R. Fleischer, M. Wahl // J. of Scheduling. – 2000. – Vol. 3. – P. 343–353.

4. Faigle, U. On the performance of on-line algorithms for partition problems / U. Faigle, W. Kern, G. Turan // Acta Cybernetica. – 1989. – Vol. 9. – P. 107–119.

5. Rudin III, J.F. Improved bounds for the online scheduling problem / J.F. Rudin III, R. Chandrasekaran // SIAM J. on Computing. – 2003. – Vol. 32. – P. 717–735.

6. Semi on-line algorithms for the partition problem / H. Kellerer [et al.] // Operations Research Letters. – 1997. – Vol. 21. – P. 235–242.

7. Azar, Y. On-line bin-stretching / Y. Azar, O. Regev // Theoretical Computer Sci. – 2001. – Vol. 268. – P. 17–41.

8. Cheng, T.C.E. Semi-on-line multiprocessor scheduling with given total processing time / T.C.E. Cheng, H. Kellerer, V. Kotov // Theoretical Computer Sci. – 2005. – Vol. 337. – P. 134–146.

9. Albers, S. Semi-Online Scheduling Revisited / S. Albers, M. Hellwig // Theoretical Computer Sci. – 2012. – Vol. 443. – P. 1–9.

Поступила 22.06.2012

[1]*Белорусский государственный университет,*
Минск, пр. Независимости, 4
e-mail: kotovvm@bsu.by

[2]*Университет Граца, Университетштрассе 15,*
А-8010, Австрия
e-mail: hans.kellerer@uni-graz.at

[3]*Университет Гренобля,*
Феликс Виоле пр., Гренобль, Франция
e-mail: nadia.brauner@grenoble-inp.fr
gerd.finke@g-scop.inpg.fr

V.M. Kotov, H. Kellerer, N. Brauner, G. Finke

AN ALGORITHM FOR SEMI ONLINE VERSIONS OF THE PROBLEM $Pm||C_{\max}$

We propose a parametric scheme for two versions of semi online multiprocessor scheduling problem to minimize makespan, with a priori known total processing time and a priori known optimal value, respectively, which provides the best known worst-case approximation for these versions.