5.96 cumulative

DESCRIPTION LINKS GRAPH AUTOMATON

Origin [1]

Constraint `cumulative(TASKS, LIMIT)`

Synonym `cumulative_max`.

Arguments

$$\begin{array}{lll} \texttt{TASKS} & : & \texttt{collection}\left(\begin{array}{l}\texttt{origin}-\texttt{dvar},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{height}-\texttt{dvar}\end{array}\right) \\ \texttt{LIMIT} & : & \texttt{int} \end{array}$$

Restrictions

`require_at_least`$(2, \texttt{TASKS}, [\texttt{origin}, \texttt{duration}, \texttt{end}])$
`required`$(\texttt{TASKS}, \texttt{height})$
$\texttt{TASKS.duration} \geq 0$
$\texttt{TASKS.origin} \leq \texttt{TASKS.end}$
$\texttt{TASKS.height} \geq 0$
$\texttt{LIMIT} \geq 0$

Purpose

Cumulative scheduling constraint or scheduling under resource constraints. Consider a set $\mathcal{T}$ of tasks described by the `TASKS` collection. The `cumulative` constraint enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. It also imposes for each task of $\mathcal{T}$ the constraint $\texttt{origin} + \texttt{duration} = \texttt{end}$.

Example

$$\left(\left\langle\begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-3 & \texttt{end}-4 & \texttt{height}-1, \\ \texttt{origin}-2 & \texttt{duration}-9 & \texttt{end}-11 & \texttt{height}-2, \\ \texttt{origin}-3 & \texttt{duration}-10 & \texttt{end}-13 & \texttt{height}-1, \\ \texttt{origin}-6 & \texttt{duration}-6 & \texttt{end}-12 & \texttt{height}-1, \\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9 & \texttt{height}-3 \end{array}\right\rangle, 8\right)$$

Figure 5.230 shows the cumulated profile associated with the example. To each task of the `cumulative` constraint, i.e. each line of the example, corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. The `cumulative` constraint holds since at each point in time we do not have a cumulated resource consumption strictly greater than the upper limit 8 enforced by the last argument of the `cumulative` constraint.

TASKS

①	o − 1	d − 3	e − 4	h − 1
②	o − 2	d − 9	e − 11	h − 2
③	o − 3	d − 10	e − 13	h − 1
④	o − 6	d − 6	e − 12	h − 1
⑤	o − 7	d − 2	e − 9	h − 3

(o for `origin`, d for `duration`, e for `end`, h for `height`)

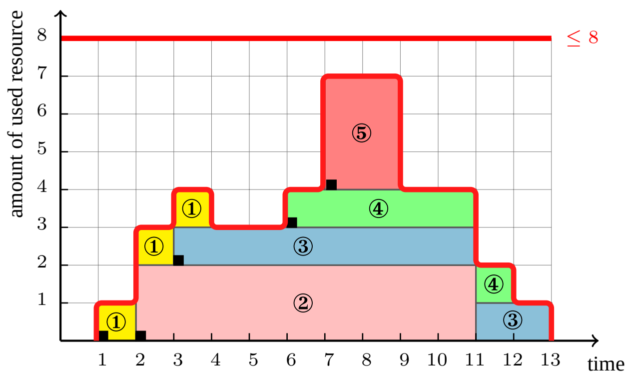


Figure 5.230: Resource consumption profile corresponding to the five tasks of the **Example** slot (note that the vertical position of a task does not really matter but is only used for displaying the contribution of a task to the resource consumption profile)

All solutions

Figure 5.231 gives all solutions to the following non ground instance of the `cumulative` constraint:
$\mathtt{O}_1 \in [1,5]$, $\mathtt{D}_1 \in [4,4]$, $\mathtt{E}_1 \in [1,9]$, $\mathtt{H}_1 \in [2,6]$,
$\mathtt{O}_2 \in [2,7]$, $\mathtt{D}_2 \in [6,6]$, $\mathtt{E}_2 \in [1,9]$, $\mathtt{H}_2 \in [3,3]$,
$\mathtt{O}_3 \in [3,6]$, $\mathtt{D}_3 \in [3,6]$, $\mathtt{E}_3 \in [1,9]$, $\mathtt{H}_3 \in [1,2]$,
$\mathtt{O}_4 \in [1,8]$, $\mathtt{D}_4 \in [2,3]$, $\mathtt{E}_4 \in [1,9]$, $\mathtt{H}_4 \in [3,4]$,
$\mathtt{cumulative}(\langle \mathtt{O}_1\ \mathtt{D}_1\ \mathtt{E}_1\ \mathtt{H}_1\ 1,\ \mathtt{O}_2\ \mathtt{D}_2\ \mathtt{E}_2\ \mathtt{H}_2\ 2,\ \mathtt{O}_3\ \mathtt{D}_3\ \mathtt{E}_3\ \mathtt{H}_3\ 3,\ \mathtt{O}_4\ \mathtt{D}_4\ \mathtt{E}_4\ \mathtt{H}_4\ 4\rangle, 5)$.

Typical

$|\mathtt{TASKS}| > 1$
$\mathtt{range}(\mathtt{TASKS.origin}) > 1$
$\mathtt{range}(\mathtt{TASKS.duration}) > 1$
$\mathtt{range}(\mathtt{TASKS.end}) > 1$
$\mathtt{range}(\mathtt{TASKS.height}) > 1$
$\mathtt{TASKS.duration} > 0$
$\mathtt{TASKS.height} > 0$
$\mathtt{LIMIT} < \mathtt{sum}(\mathtt{TASKS.height})$

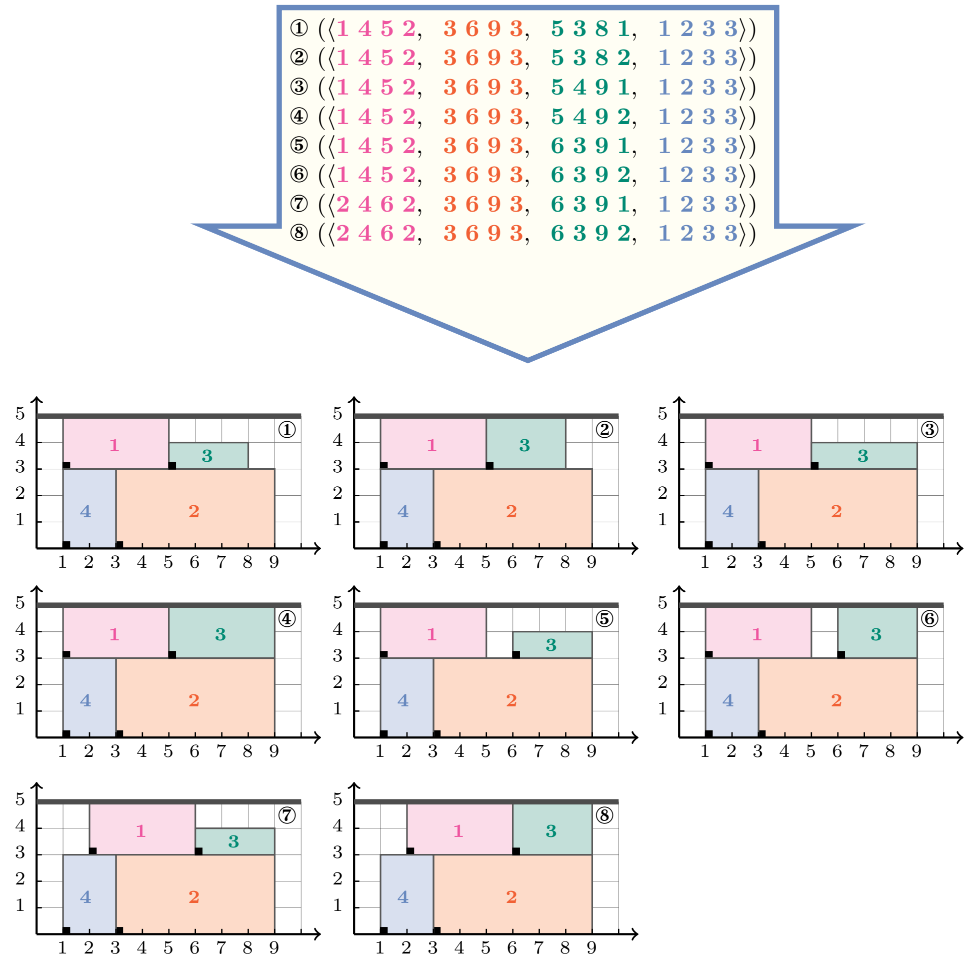


Figure 5.231: All solutions corresponding to the non ground example of the `cumulative` constraint of the **All solutions** slot

Symmetries

- Items of `TASKS` are permutable.
- `TASKS.duration` can be decreased to any value ≥ 0.
- `TASKS.height` can be decreased to any value ≥ 0.
- One and the same constant can be added to the `origin` and `end` attributes of all items of `TASKS`.
- `LIMIT` can be increased.

Arg. properties

Contractible wrt. `TASKS`.

Usage

The `cumulative` constraint occurs in most resource scheduling problems where one has to deal with renewable and/or non-renewable resources:

- *Renewable resources* typically correspond to machines or persons, and tasks require such resources during all their execution (i.e. a resource starts to be used at the beginning of the task and is released at the end of the task). This means that, once a task has finished its work, the resource it was using is available for other tasks. Tasks are defined by their *origin*, *duration*, *end* and *resource consumption* and can not be interrupted. When the duration and resource consumption are not fixed tasks can be defined by their *load*, i.e. the product of their duration and resource consumption. To express the dependency between a non-fixed duration/resource consumption of a task with another decision variable (e.g., to state that the duration of a task depends on its start) one can use the `element` constraint where the decision variable corresponds to the index argument of the `element` constraint.
- *Non renewable resources* typically correspond to stock or money, i.e., resources that do not come back when a task finishes to use them. In this context the `cumulative` constraint is used for modelling producer-consumer problems, i.e. problems where a first set of tasks produces a non-renewable resource, while a second set of tasks consumes this resource so that a limit on the minimum or the maximum stock at each instant is imposed.

The `cumulative` constraint is also used as a necessary condition for non-overlapping rectangles (see the `diffn` constraint).

Remark

In the original `cumulative` constraint of **CHIP** the `LIMIT` parameter was a domain variable corresponding to the *maximum peak of the resource consumption profile*. Given a fixed time frame, this variable could be used as a cost in order to directly minimise the maximum resource consumption peak. Fixing this variable is potentially dangerous since it imposes the maximum peak *to be equal to a given target value*.

Some systems like Ilog CP Optimizer also assume that a zero-duration task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is greater than or equal to i. Under this definition, the height of a zero-duration task is also taken into account in the resource consumption profile.

Note that the concept of cumulative is *different* from the concept of rectangles non-overlapping even though, most of the time, each task of a ground solution to a `cumulative` constraint is simply drawn as a single rectangle. As illustrated by Figure 5.283, this is in fact not always possible (i.e., some rectangles may need to be broken apart). In fact the `cumulative` constraint is only a necessary condition for rectangles non-overlapping (see Figure 5.282 and the corresponding explanation in the **Algorithm** slot of the `diffn` constraint).

In **MiniZinc** (`http://www.minizinc.org/`) the tasks of a `cumulative` constraint have no `end` attribute.

Algorithm

The first filtering algorithms were related to the notion of compulsory part of a task [250]. They compute a cumulated resource profile of all the compulsory parts of the tasks and prune the origins of the tasks with respect to this profile in order to not exceed the resource capacity. These methods are sometimes called *time tabling*. Even though these methods are quite local, i.e., a task has a non-empty compulsory part only when the difference between its latest start and its earliest start is strictly less than its duration, it scales well and is therefore widely used. Later on, more global algorithms[5] based on the resource consumption of

[5] Even though these more global algorithms usually can prune more early in the search tree, these algorithms do not catch all deductions derived from the cumulated resource profile of compulsory parts.

the tasks on specific intervals were introduced [154, 103, 265]. A popular variant, called *edge finding*, considers only specific intervals [283]. An efficient implementation of edge finding in $O(kn \log n)$, where k is the number of distinct task heights and n is the number of tasks, based on a specific data structure, so called a *cumulative Φ-tree* [435], is provided in [434]. When the number of distinct task heights k is not small, a usually almost faster implementation in $O(n^2)$ is described in [230]. A $O(n^2 \log n)$ filtering algorithm based on tasks that can not be the earliest (or not be the latest) is described in [375].

Within the context of linear programming, the reference [216] provides a relaxation of the `cumulative` constraint.

A necessary condition for the `cumulative` constraint is obtained by stating a `disjunctive` constraint on a subset of tasks $\mathcal{T}$ such that, for each pair of tasks of $\mathcal{T}$, the sum of the two corresponding minimum heights is strictly greater than `LIMIT`. This can be done by applying the following procedure:

- Let h be the smallest minimum height strictly greater than $\lfloor \frac{\mathtt{LIMIT}}{2} \rfloor$ of the tasks of the `cumulative` constraint. If no such task exists then the procedure is stopped without stating any `disjunctive` constraint.
- Let $\mathcal{T}_h$ denote the set of tasks of the `cumulative` constraint for which the minimum height is greater than or equal to h. By construction, the tasks of $\mathcal{T}_h$ cannot overlap. But we can possibly add one more task as shown by the next step.
- When it exists, we can add one task that does not belong to $\mathcal{T}_h$ and such that its minimum height is strictly greater than $\mathtt{LIMIT} - h$. Again, by construction, this task cannot overlap all the tasks of $\mathcal{T}_h$.

When the tasks are involved in several `cumulative` constraints more sophisticated methods are available for extracting `disjunctive` constraints [17, 16].

In the context where, both the duration and height of all the tasks are fixed, [37] provides two kinds of additional filtering algorithms that are especially useful when the slack σ (i.e., the difference between the available space and the sum of the surfaces of the tasks) is very small:

- The first one introduces bounds for the so called *cumulative longest hole problem*. Given an integer ϵ that does not exceed the resource limit, and a subset of tasks $\mathcal{T}'$ for which the resource consumption is a most ϵ, the *cumulative longest hole problem* is to find the largest integer $lmax_\sigma^\epsilon(\mathcal{T}')$ such that there is a cumulative placement of maximum height ϵ involving a subset of tasks of $\mathcal{T}'$ where, on one interval $[i, i + lmax_\sigma^\epsilon(\mathcal{T}') - 1]$ of the cumulative profile, the area of the empty space does not exceed σ.
- The second one used dynamic programming for filtering so called *balancing knapsack constraints*. When the slack is 0, such constraints express that the total height of tasks ending at instant i must equal the total height of tasks starting at instant i. Such constraints can be generalised to non-zero slack.

Systems `cumulativeMax` in **Choco**, `cumulative` in **Gecode**, `cumulative` in **JaCoP**, `cumulative` in **MiniZinc**, `cumulative` in **SICStus**.

See also **assignment dimension added:** `coloured_cumulatives` *(sum of* `task heights` *replaced by number of distinct* `colours`*, assignment dimension added),* `cumulatives` *(negative* `heights` *allowed and assignment dimension added).*

common keyword: calendar *(scheduling constraint)*, coloured_cumulative *(resource constraint, sum of* task heights *replaced by number of distinct values)*, coloured_cumulatives *(resource constraint)*, cumulative_convex *(resource constraint,* task *defined by a set of* points*)*, cumulative_product *(resource constraint, sum of* task heights *replaced by product of* task heights*)*, cumulative_with_level_of_priority *(resource constraint, a* cumulative *constraint for each set of* tasks *having a priority less than or equal to a given threshold)*.

generalisation: cumulative_two_d *(*task *replaced by* rectangle *with a* height*)*.

implied by: diffn *(*cumulative *is a neccessary condition for each dimension of the* diffn *constraint)*.

related: lex_chain_less, lex_chain_lesseq *(lexicographic ordering on the origins of* tasks, rectangles, *...)*, ordered_global_cardinality *(controlling the shape of the cumulative profile for breaking symmetry)*.

soft variant: soft_cumulative.

specialisation: atmost *(*task *replaced by* variable*)*, bin_packing *(all* tasks *have a* duration *of* 1 *and a fixed* height*)*, disjunctive *(all* tasks *have a* height *of* 1*)*, multi_inter_distance *(all* tasks *have the same* duration *equal to* DIST *and the same* height *equal to* 1*)*.

used in graph description: sum_ctr.

Keywords

characteristic of a constraint: core, automaton, automaton with array of counters.

complexity: sequencing with release times and deadlines.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: linear programming, dynamic programming, compulsory part, cumulative longest hole problems, Phi-tree.

modelling: zero-duration task.

problems: producer-consumer.

puzzles: squared squares.

Cond. implications

$\texttt{cumulative}(\texttt{TASKS}, \texttt{LIMIT})$
with $\texttt{TASKS.height} > 0$
implies $\texttt{coloured\_cumulative}(\texttt{TASKS} : \texttt{TASKS}, \texttt{LIMIT} : \texttt{LIMIT})$.

Arc input(s)	TASKS
Arc generator	$SELF \mapsto$ collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin + tasks.duration = tasks.end
Graph property(ies)	**NARC** = |TASKS|

Arc input(s)	TASKS TASKS
Arc generator	$PRODUCT \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.duration > 0 • tasks2.origin ≤ tasks1.origin • tasks1.origin < tasks2.end
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP
Sets	SUCC ↦ [source, variables − col (VARIABLES−collection(var−dvar), [item(var − TASKS.height)])]
Constraint(s) on sets	sum_ctr(variables, ≤, LIMIT)

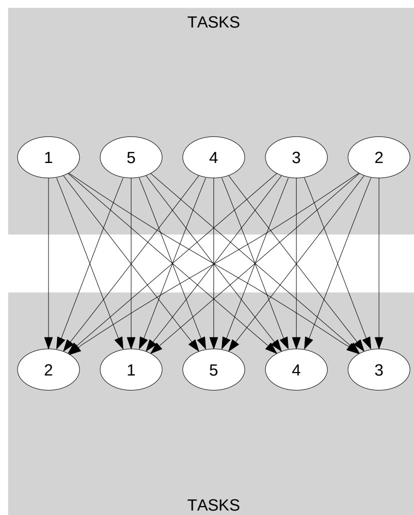
Graph model

The first graph constraint forces for each task the link between its origin, its duration and its end. The second graph constraint makes sure, for each time point t corresponding to the start of a task, that the cumulated heights of the tasks that overlap t does not exceed the limit of the resource.

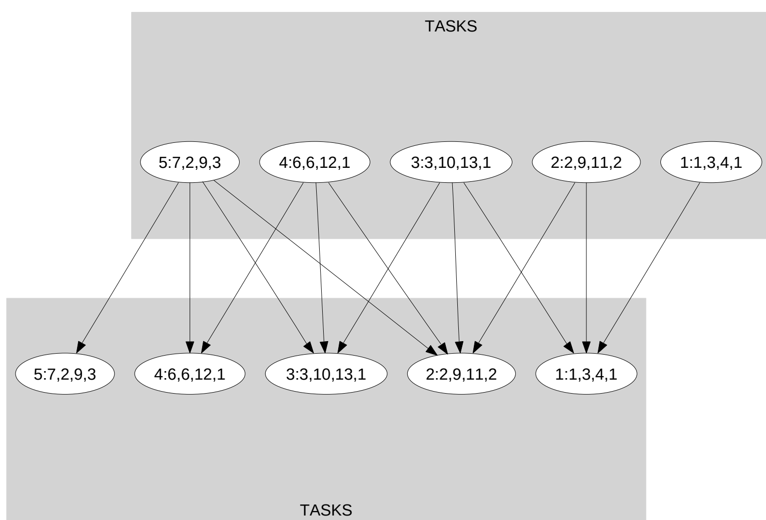
Parts (A) and (B) of Figure 5.232 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The `cumulative` constraint holds since for each successor set $\mathcal{S}$ of the final graph the sum of the heights of the tasks in $\mathcal{S}$ does not exceed the limit $\texttt{LIMIT} = 8$.

Signature

Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite **NARC** = |TASKS| to **NARC** ≥ |TASKS|. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.



(A)



(B)

Figure 5.232: Initial and final graph of the `cumulative` constraint

Automaton Figure 5.96 depicts the automaton associated with the `cumulative` constraint. To each item of the collection `TASKS` corresponds a signature variable S_i that is equal to 1.

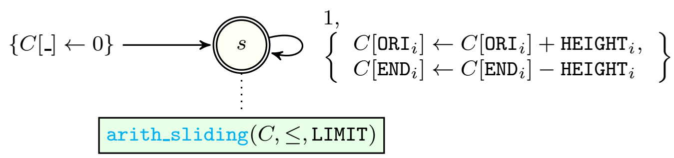


Figure 5.233: Automaton of the `cumulative` constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint `cumulative`$(\langle 1\ 2\ 3\ 3, 2\ 2\ 4\ 2, 4\ 1\ 5\ 1\rangle, 4)$ hold?

B. Does the constraint `cumulative`$(\langle 1\ 2\ 3\ 1, 4\ 1\ 5\ 2\rangle, 1)$ hold?

C. Does the constraint `cumulative`$(\langle 1\ 2\ 3\ 0, 1\ 2\ 3\ 4, 4\ 1\ 6\ 1\rangle, 4)$ hold?

[a] Hint: go back to the definition of `cumulative`.

EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\left\{ \begin{array}{l} O_1 \in [1,9],\ O_2 \in [1,9],\ O_3 \in [1,9],\ O_4 \in [1,9], \\ E_1 \in [1,8],\ E_2 \in [1,8],\ E_3 \in [1,8],\ E_4 \in [1,8], \\ \mathtt{cumulative}(\langle O_1\ 1\ E_1\ 1, O_2\ 2\ E_2\ 2, O_3\ 3\ E_3\ 5, O_4\ 4\ E_4\ 7\rangle, 7). \end{array} \right.$$

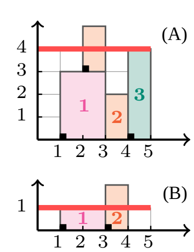
[a] Hint: reason first on the two highest tasks and then on the other tasks.

SOLUTION TO EXERCISE 1

A. *No, since the first and second tasks overlap at time point* 2 *and use up to* $3+2$ *resource units which exceeds the resource capacity* 4.

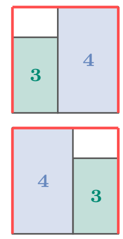
B. *No, since the second task uses* 2 *resource units, while the resource capacity is* 1.

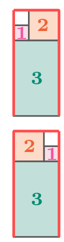
C. *No, since for the third task the origin plus the duration is different from the end* $(4+1 \neq 6)$.



SOLUTION TO EXERCISE 2
(*nested disjunctions*)

1. *Since we have a resource limit of* 7 *the third task (of height* 5*) cannot overlap the fourth task (of height* 7*). Since there is no slack on the time axis (i.e., the difference between the latest end of the third and fourth tasks and their earliest start is equal to the sum of their durations,* $8 - 1 = 3 + 4$*), this leads to the two configurations shown on the right.*
2. *Since there is no available space on top of the fourth task, the first and second tasks have to be put on top of the third task. Since on top of the third task we only have a capacity of* 2 *the first and second tasks cannot overlap. Since there is no remaining slack on the time axis this leads to the two configurations shown on the right.*
3. *Combining the two previous observations together leads to the four solutions shown below.*





the four solutions

$\langle O_1 D_1 E_1 H_1, O_2 D_2 E_2 H_2, O_3 D_3 E_3 H_3, O_4 D_4 E_4 H_4 \rangle$

① $(\langle 1\ 1\ 2\ 1,\ \ 2\ 2\ 4\ 2,\ \ 1\ 3\ 4\ 5,\ \ 4\ 4\ 8\ 7 \rangle)$
② $(\langle 3\ 1\ 4\ 1,\ \ 1\ 2\ 3\ 2,\ \ 1\ 3\ 4\ 5,\ \ 4\ 4\ 8\ 7 \rangle)$
③ $(\langle 5\ 1\ 6\ 1,\ \ 6\ 2\ 8\ 2,\ \ 5\ 3\ 8\ 5,\ \ 1\ 4\ 5\ 7 \rangle)$
④ $(\langle 7\ 1\ 8\ 1,\ \ 5\ 2\ 7\ 2,\ \ 5\ 3\ 8\ 5,\ \ 1\ 4\ 5\ 7 \rangle)$