Планирование портовых операций, сведение к minizinc-у.

## Основные определения:

**Движущийся объект/судно** – объект, способный перемещаться между локациями – транспортный корабль, буксир или оборудование грузообработки. Если объект в принципе не способен перемещаться (например, оборудование “насос”), назовём его судном без операций перемещения.

**Субъект/главный объект операции** – движущийся объект, которого перемещают, швартуют/отшвартовывают или с которым выполняют операцию грузообработки.

**Операция перемещения** – у главного объекта меняется локация – операции перемещения между причалами, швартовка и отшвартовка.

## Основные константы:

**n\_intervals** : int – горизонт планирования, количество элементарных временных интервалов.

**n\_operations :** int – количество всех операций.

**n\_locations :** int – количество локаций, чётные локации соответствуют состояниям “отшвартовано”, чётная + 1 – пришвартовано на той физической локации, что и чётная.

**n\_moving\_obj :** int – количество движущихся объектов.

## Описание:

### Сведе́ния:

На текущий момент реализованные сведения имеют много общего, так что далее все описанные техники, структурные части и оптимизации использованы во всех сведениях, если не указано иного.

Реализованные сведения:

#### **С неделимыми операциями**

Операции грузообработки всегда имеют фиксированную интенсивность.

Достоинства – наиболее быстрое нахождение ответа.

Недостатки – не полностью поддерживает модель; когда требуются нецелые операции не находит решение, либо находит, но не оптимальное (пример задачи - *tests/with\_typing/Case1.tipp*).

Реализация – *constraints\_2.mzn*

#### **С вариативной интенсивностью операций грузообработки**

Операции грузообработки имею произвольную интенсивность от 1 до номинального значения.

Достоинства – полное соответствие модели; при максимальной номинальной интенсивности операций грузообработки не больше 2-ух по скорости нахождения ответа сравнима с сведением с неделимыми операциями.

Недостатки – время решения растёт экспоненциально с ростом номинальной интенсивности. При интенсивности 100 решения не работают за разумное время.

Реализация - *constraints/conversion\_2\_with\_partial\_cargo\_operations.mzn*

#### **С жадным выбором интенсивности**

Интенсивности операций задаются вручную от остальных переменных с помощью простой эвристики.

Достоинства – на “реальных” тестах находит оптимальное решение, даже когда требуются нецелые операции.

Недостатки – по скорости решения уступает сведению с неделимыми операциями; не полностью соответствует модели – из-за эвристики оптимальное решение может быть не найдено (пример задачи - *tests/with\_typing/anti\_greedy.tipp*). В текущей реализации существуют несоответствия с моделью сведения с нецелыми операциями (подробнее в описании).

Реализация - *constraints/conversion\_2\_greedy.mzn* и *constraints/conversion\_2\_greedy.mzn*

### Подготовка:

Уберём типизацию по основному субъекту операции – заменим каждую операцию на её копии, где вместо типа указан конкретный объект (по количеству подходящих объектов).

Также, для упрощения реализации, в операциях бункеровки уберём типизацию по бункеровщику.

### Статус операции:

**op\_status** : array [1..n\_operations, 0..(n\_intervals + 1)] of var bool – статус операции – true – выполняется в текущий интервал, false – не выполняется. В крайних интервалах – 0 и n\_intervals + 1 ничего не выполняется. Основные переменные, остальное так или иначе будет завязано на них.

### Основные параметры-переменные движущегося объекта:

**current\_moving\_operation** : array [1..n\_moving\_obj, 0..(n\_intervals + 1)] of var 0..n\_operations

– номер операции перемещения, в которой участвует объект, или 0, если в таковой операции объект не участвует.

**moving\_op\_of\_obj :** array [1..n\_moving\_obj, 1..n\_operations] of bool – вспомогательные константы,операции перемещения, которые могут затрагивать данный объект – true – могут; false – не могут.

##### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Текущая операция лежит в множестве тех, которые затрагивают данный объект. |
| 2 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 3 |  | (current\_moving\_operation[obj, t] != 0) -> moving\_op\_of\_obj[obj, current\_moving\_operation[obj, t]] |
| 4 |  | ); |
| 5 |  | % Если операция, затрагивающая данный объект в качестве главного, выполняется - то именно она текущая операция перемещения для этого объекта. |
| 6 |  | constraint forall (op in 1..n\_operations, t in 1..n\_intervals where is\_moving\_operation[op]) ( |
| 7 |  | op\_status[op, t] -> (current\_moving\_operation[main\_obj\_of\_operation[op], t] = op) |
| 8 |  | ); |
| 9 |  | % Связь current\_moving\_operation c op\_status. |
| 10 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 11 |  | (current\_moving\_operation[obj, t] != 0) -> (op\_status[current\_moving\_operation[obj, t], t]) |
| 12 |  | ); |

**m\_obj\_loc** : array [1..n\_moving\_obj, 0..(n\_intervals + 1)] of var 1..n\_locations

– локация объекта, в которой он находится, или в которую он перемещается перед началом интервала.

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Направление движения/местоположение объекта может измениться только если перед этим началась операция перемещения, и он не вспомогательный при операции швартовки. |
| 2 |  | constraint forall (obj in 1..n\_moving\_obj, t in 2..(n\_intervals + 1)) ( |
| 3 |  | ((current\_moving\_operation[obj, t - 1] != current\_moving\_operation[obj, t - 2]) /\ |
| 4 |  | (current\_moving\_operation[obj, t - 1] != 0) /\ |
| 5 |  | ((obj = main\_obj\_of\_operation[current\_moving\_operation[obj, t - 1]]) |
| 6 |  | \/ |
| 7 |  | (not is\_mooring\_op[current\_moving\_operation[obj, t - 1]]) |
| 8 |  | ) |
| 9 |  | ) -> (m\_obj\_loc[obj, t] = operations\_destination[current\_moving\_operation[obj, t - 1]]) |
| 10 |  | ); |
| 11 |  | constraint forall (obj in 1..n\_moving\_obj, t in 2..(n\_intervals + 1)) ( |
| 12 |  | ((current\_moving\_operation[obj, t - 1] = current\_moving\_operation[obj, t - 2]) \/ |
| 13 |  | (current\_moving\_operation[obj, t - 1] = 0) \/ |
| 14 |  | ((obj != main\_obj\_of\_operation[current\_moving\_operation[obj, t - 1]]) |
| 15 |  | /\ |
| 16 |  | ( is\_mooring\_op[current\_moving\_operation[obj, t - 1]]) |
| 17 |  | ) |
| 18 |  | ) -> (m\_obj\_loc[obj, t] = m\_obj\_loc[obj, t - 1]) |
| 19 |  | ); |

### Участие объектов в качестве ресурсов, связь с основными переменными:

Основная идея – на каждую операцию, на каждый тип ресурсов, что встречается в ресурсах данной операции (и, разумеется, на каждый интервал) завести счётчик объектов этого типа (**resources\_counter**), которые участвуют как ресурсы(**participation\_as\_resource**) в данной операции.

Если операция выполняется, то все счётчики этой операции должны равняться нужному количеству ресурсов соответствующего типа для её запуска.

Так же обяжем ресурсы не меняться во время всей операции – существенно для поддержания **current\_moving\_operation** при операции перемещения; при операции погрузки – не влияет на наличие решения, может сокращать их множество.

Для простоты реализации все счётчики имеют сквозную нумерацию, информация о них идёт в вспомогательных массивах.

#### Константы:

**operations\_that\_used\_obj\_as\_resource** : array [1..n\_moving\_obj] of set of 1..n\_operations – операции, которые могут использовать данный объект как ресурс.

**is\_moving\_operation** : array [0..n\_operations] of bool – является ли операция операцией перемещения.

**n\_resources\_types** : int - количество различных типов ресурсов.

**objects\_of\_type** : array [1..n\_resources\_types] of set of 1..n\_moving\_obj - все объекты конкретного типа.

**n\_resources\_counters** : int - количество счётчиков ресурсов.

**counter\_type** : array [1..n\_resources\_counters] of 1..n\_resources\_types - тип ресурсов, за которым следит счётчик.

**operation\_of\_counter** : array [1..n\_resources\_counters] of 1..n\_operations - операция, которой принадлежит данный счётчик.

**required\_counter\_values** : array [1..n\_resources\_counters] of int - необходимые значения на счётчиках ресурсов для выполнения операции.

**counters\_of\_operation** : array [1..n\_operations] of set of 1..n\_resources\_counters - счётчики, которые относятся к данной операции.

#### Переменные:

**participation\_as\_resource** : array [1..n\_moving\_obj, 0..(n\_intervals + 1)] of var 0..n\_operations – операция, в которой объект участвует как ресурс, или “0”, если в таковой не участвует.

##### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Граничные значения. |
| 2 |  | constraint forall (obj in 1..n\_moving\_obj) (participation\_as\_resource[obj, 0] = 0); |
| 3 |  | constraint forall (obj in 1..n\_moving\_obj) (participation\_as\_resource[obj, n\_intervals + 1] = 0); |
| 4 |  |  |
| 5 |  | % Только те операции, которые затрагивают данный объект. |
| 6 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 7 |  | (participation\_as\_resource[obj, t] != 0) |
| 8 |  | -> participation\_as\_resource[obj, t] in operations\_that\_used\_obj\_as\_resource[obj] |
| 9 |  | ); |
| 10 |  |  |
| 11 |  | % Связь с текущими операциями перемещения. |
| 12 |  | % Если объект задействован в операции перемещения, которая использует его как ресурс, |
| 13 |  | % то participation\_as\_resource должен указывать на эту операцию. |
| 14 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 15 |  | (current\_moving\_operation[obj, t] in operations\_that\_used\_obj\_as\_resource[obj]) |
| 16 |  | -> |
| 17 |  | (participation\_as\_resource[obj, t] = current\_moving\_operation[obj, t]) |
| 18 |  | ); |
| 19 |  | % Если объект участвует как ресурс в операции перемещения, то это согласованно с current\_moving\_operation. |
| 20 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 21 |  | ((participation\_as\_resource[obj, t] != 0) /\ |
| 22 |  | (is\_moving\_operation[participation\_as\_resource[obj, t]]) |
| 23 |  | ) -> (participation\_as\_resource[obj, t] = current\_moving\_operation[obj, t]) |
| 24 |  | ); |
| 25 |  |  |
| 26 |  | % { Объект участвует где-то в качестве ресурса - соответствующая операция обязана быть активной. } |
| 27 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 28 |  | (participation\_as\_resource[obj, t] != 0) -> op\_status[participation\_as\_resource[obj, t], t] |
| 29 |  | ); |
| 30 |  |  |
| 31 |  | % От начала операции и до конца её ресурсы не могут измениться (в том числе и для погрузки). |
| 32 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..(n\_intervals - 1)) ( |
| 33 |  | ((participation\_as\_resource[obj, t] != 0) /\ |
| 34 |  | (op\_status[participation\_as\_resource[obj, t], t + 1]) |
| 35 |  | ) -> (participation\_as\_resource[obj, t + 1] = participation\_as\_resource[obj, t]) |
| 36 |  | ); |

**resources\_counter** : array [1..n\_resources\_counters , 1..n\_intervals] of var int – счётчик объектов определённого типа, участвующих в операции, к которой относится этот счётчик.

##### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Участие всех необходимых ресурсов в операции. |
| 2 |  | constraint forall (t in 1..n\_intervals, op in 1..n\_operations, counter in counters\_of\_operation[op]) ( |
| 3 |  | op\_status[op, t] -> (resources\_counter[counter, t] = required\_counter\_values[counter]) |
| 4 |  | ); |
| 5 |  | % Определение resources\_counter. |
| 6 |  | constraint forall (counter in 1..n\_resources\_counters, t in 1..n\_intervals) ( |
| 7 |  | resources\_counter[counter, t] = sum (obj in objects\_of\_type[counter\_type[counter]]) ( |
| 8 |  | participation\_as\_resource[obj, t] = operation\_of\_counter[counter] |
| 9 |  | ) |
| 10 |  | ); |

### Наличие на месте и готовность субъектов операции к её началу:

В момент начала операции её субъект должен:

* Быть в нужной локации.
* Если это операция перемещения, то субъект не участвует в других операциях – (для других операций перемещения достигается определением **current\_moving\_operation**).

#### Константы:

**related\_cargo\_op** : array [1..n\_moving\_obj] of set of 1..n\_operations – операции грузообработки, которые используют объект как главный или в качестве бункеровщика.

#### Переменные:

**is\_involved\_in\_cargo\_op** : array [1..n\_moving\_obj, 1..n\_intervals] of var bool – участвует ли объект в этот момент в операции грузообработки.

##### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Определение is\_involved\_in\_cargo\_op. |
| 2 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 3 |  | is\_involved\_in\_cargo\_op[obj, t] = |
| 4 |  | (exists (op in related\_cargo\_op[obj]) (op\_status[op, t])) |
| 5 |  | \/ |
| 6 |  | ((participation\_as\_resource[obj, t] != 0) /\ (not is\_moving\_operation[participation\_as\_resource[obj, t]])) |
| 7 |  | ); |

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Операции перемещения исключают операции грузообработки и наоборот. |
| 2 |  | constraint forall (obj in 1..n\_moving\_obj, t in 1..n\_intervals) ( |
| 3 |  | is\_involved\_in\_cargo\_op[obj, t] -> (current\_moving\_operation[obj, t] = 0) |
| 4 |  | ); |
| 5 |  | % Наличие главных объектов на месте. |
| 6 |  | constraint forall (op in 1..n\_operations, |
| 7 |  | t in 1..n\_intervals) ( |
| 8 |  | let {1..n\_moving\_obj : obj = main\_obj\_of\_operation[op]} in |
| 9 |  | op\_start[op, t] -> (m\_obj\_loc[obj, t] == main\_obj\_start\_loc[op]) |
| 10 |  | ); |

### Наличие на месте и готовность ресурсов операции к её началу:

Идейно – то же самое, что и для субъектов. Заметим, что стартовые локации субъекта и ресурсов в общем случае различаются (например, при отшвартовке).

#### Константы:

**operations\_resources\_start\_loc** : array [1..n\_resources\_counters] of 1..n\_locations – стартовая локация ресурса, необходимая для начала операции.

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Наличие на месте всех ресурсов. |
| 2 |  | constraint forall ( |
| 3 |  | op in 1..n\_operations, |
| 4 |  | t in 1..n\_intervals, |
| 5 |  | counter in counters\_of\_operation[op], |
| 6 |  | obj in objects\_of\_type[counter\_type[counter]]) ( |
| 7 |  | ((participation\_as\_resource[obj, t] = op) /\ (op\_start[op, t]) |
| 8 |  | ) -> (m\_obj\_loc[obj, t] == operations\_resources\_start\_loc[counter]) |
| 9 |  | ); |

### Непрерывность операций:

Ограничение – для каждого интервала – если операция началась, то она длится нужное количество интервалов, а после – обязательно прерывается.

#### Константы:

**operations\_duration** : array [1..n\_operations] of int - длительность операции в элементарных интервалах.

**is\_continuous\_operation** : array [1..n\_operations] of bool – является ли данная операция непрерывной.

#### Переменные:

**op\_start** : array [1..n\_operations, 0..(n\_intervals + 1)] of var bool – началась ли операция в текущий интервал - *op\_start[i, j] = (op\_status[i, j] /\ not op\_status[i, j - 1]).*

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | constraint forall (i in 1..n\_operations, len = operations\_duration[i] where is\_continuous\_operation[i]) ( |
| 2 |  | (forall (j in 1..(n\_intervals - len + 1)) ( |
| 3 |  | (op\_start[i, j] == 1) -> ( |
| 4 |  | (forall (k in 1..(len - 1)) (op\_status[i, j + k])) |
| 5 |  | /\ |
| 6 |  | (not op\_status[i, j + len]) |
| 7 |  | ) |
| 8 |  | )) /\ |
| 9 |  | (forall (j in (n\_intervals - len + 2)..(n\_intervals + 1)) (op\_start[i, j] == false)) |
| 10 |  | ); |

### Грузообработка при неделимых операциях (фиксированной интенсивности):

Необходимо поддержать следующие условия:

* Неотрицательность объёма груза и не превышение максимального размера хранилища.
* Нужные граничные значения.
* Текущий объём груза = предыдущий + величина потока груза в данный интервал + сумма всех изменений объёма от активных операций, связанных с этим хранилищем.

Заметим, у модели в текущем виде каждый груз имеет свой отсек, грузы по определению не могут смешиваться или занимать общий объём.

#### Константы:

**n\_cargo\_types** : int – количество типов грузов.

**n\_all\_storage\_sections** : int – количество всех секций хранилищ. Не учитывает нулевую, фиктивную секцию. К примеру, если судно может хранить “Груз 1” объёмом “Volume 1”, “Груз 2” объёмом “Volume 2” и “Груз 3” объёмом “Volume 3”, то это судно породит 3 соответствующих секции.

**max\_storage\_vol** : array [0.. n\_all\_storage\_sections] of int – максимальный объём хранилищ.

**initial\_storage\_vol** : array [0.. n\_all\_storage\_sections] of int – начальный объём хранилищ.

**final\_storage\_vol** : array [1..n\_obj\_with\_storage] of int – конечный объём хранилищ.

**cargo\_flows** : array [0..n\_all\_storage\_sections, 0..(n\_intervals + 1), 1..n\_cargo\_types] of int – потоки грузов.

**n\_loading\_op** : int – количество операций грузообработки.

**loading\_op\_n** : array [1..n\_loading\_op] of 1..n\_operations – номера операций грузообработки среди общего списка операций.

**involved\_operations** : array [1..n\_all\_storage\_sections] of set of 1..n\_loading\_op – операции, которые изменяют количество этого груза в данном хранилище.

**loading\_op\_delta\_of\_main\_obj** : array [1..n\_operations] of int – изменение хранилища главного объекта операции грузообработки. Если данная операция не грузообработка, тогда здесь и в дальнейшем будут стоять произвольные подходящие значения.

**operations\_main\_stor** : array [1..n\_operations] of 1..n\_obj\_with\_storage – номер хранилища субъекта операции грузообработки.

**operations\_secondary\_stor** : array [1..n\_operations] of 1..n\_obj\_with\_storage – номер другого хранилища (не субъекта) участвующего в операции грузообработки.

**operations\_cargo\_t** : array [1..n\_operations] of 1..n\_cargo\_types – тип груза операции грузообработки.

#### Переменные:

**storage\_used\_volume** : array [0..n\_all\_storage\_sections, 0..(n\_intervals + 1)] of var int – использованный объём хранилища по каждому типу груза на конец интервала.

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Максимальный объём. |
| 2 |  | constraint forall (storage in 1..n\_all\_storage\_sections, t in 0..(n\_intervals + 1)) ( |
| 3 |  | (storage\_used\_volume[storage, t]) <= max\_storage\_vol[storage] |
| 4 |  | ); |
| 5 |  | % Неотрицательность объёма. |
| 6 |  | constraint forall (storage in 1..n\_all\_storage\_sections, t in 0..(n\_intervals + 1)) ( |
| 7 |  | 0 <= storage\_used\_volume[storage, t] |
| 8 |  | ); |
| 9 |  | % Граничные значения |
| 10 |  | constraint forall (storage in 1..n\_all\_storage\_sections) ( % Initial values. |
| 11 |  | storage\_used\_volume[storage, 0] = initial\_storage\_vol[storage] |
| 12 |  | ); |
| 13 |  | constraint forall (storage in 1..n\_all\_storage\_sections % Final values. |
| 14 |  | where final\_storage\_vol[storage] >= 0) ( |
| 15 |  | storage\_used\_volume[storage, n\_intervals + 1] = final\_storage\_vol[storage] |
| 16 |  | ); |
| 17 |  | % Изменение грузов в хранилищах. |
| 18 |  | constraint forall (storage in 1..n\_all\_storage\_sections, t in 1..(n\_intervals + 1)) ( |
| 19 |  | storage\_used\_volume[storage, t] = ( |
| 20 |  | storage\_used\_volume[storage, t - 1] + |
| 21 |  | cargo\_flows[storage, t] + |
| 22 |  | sum (inv\_op in involved\_operations[storage]) ( |
| 23 |  | loading\_op\_delta[inv\_op] \* |
| 24 |  | op\_status[loading\_op\_n[inv\_op], t] |
| 25 |  | ) |
| 26 |  | ) |
| 27 |  | ); |

### Грузообработка с вариативной интенсивностью операций:

Основа та же, что и при фиксированных операциях, за исключением изменения грузов в хранилищах. Добавляются:

#### Константы:

**loading\_op\_local\_direction** : array [1..n\_loading\_op] of {1, -1} – направление операции – прибавляется или убывает груз в соответствующее хранилище.

**loading\_op\_abs\_delta** : array [0..n\_operations] of int – номинальная интенсивность операций грузообработки взятая по модулю.

#### Переменные:

**cargo\_op\_intensity** : array [0..n\_operations, 0..(n\_intervals + 1)] of var int – интенсивность операции грузообработки (реальная).

##### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Фиктивная операция. |
| 2 |  | constraint forall (t in 0..(n\_intervals + 1)) (cargo\_op\_intensity[0, t] = 0); |
| 3 |  | % Краевые значения. |
| 4 |  | constraint forall (op in 1..n\_operations, t in 0..(n\_intervals + 1)) (cargo\_op\_intensity[op, t] >= 0); |
| 5 |  | constraint forall (op in 1..n\_operations, t in 0..(n\_intervals + 1)) (cargo\_op\_intensity[op, t] <= loading\_op\_abs\_delta[op]); |
| 6 |  | % Связь с операциями грузообработки. |
| 7 |  | constraint forall (op in 1..n\_operations, t in 0..(n\_intervals + 1) where not is\_moving\_operation[op]) ( |
| 8 |  | (cargo\_op\_intensity[op, t] > 0) -> op\_status[op, t] |
| 9 |  | ); |
| 10 |  | constraint forall (op in 1..n\_operations, t in 0..(n\_intervals + 1) where is\_moving\_operation[op]) ( |
| 11 |  | cargo\_op\_intensity[op, t] = 0 |
| 12 |  | ); |
| 13 |  | constraint forall (op in 1..n\_operations, t in 0..(n\_intervals + 1) where not is\_moving\_operation[op]) ( |
| 14 |  | op\_status[op, t] -> (cargo\_op\_intensity[op, t] > 0) |
| 15 |  | ); |

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | constraint forall (storage in 1..n\_all\_storage\_sections, t in 1..(n\_intervals + 1)) ( |
| 2 |  | storage\_used\_volume[storage, t] = ( |
| 3 |  | storage\_used\_volume[storage, t - 1] + |
| 4 |  | cargo\_flows[storage, t] + |
| 5 |  | sum (inv\_op in involved\_operations[storage]) ( |
| 6 |  | cargo\_op\_intensity[loading\_op\_n[inv\_op], t] \* |
| 7 |  | loading\_op\_local\_direction[inv\_op] |
| 8 |  | ) |
| 9 |  | ) |
| 10 |  | ); |

### Грузообработка с эвристическим выбором интенсивности:

Как следует из названия, основная идея этого сведения – помочь solver-у, вручную рассчитав интенсивность грузовых операций.

За основу берём грузообработку с вариативной интенсивностью, удалив для начала cargo\_op\_intensityи ограничения, связанные с ней.

#### Переменные:

**nominal\_cargo\_value** : array [0..n\_operations, 0..(n\_intervals + 1)] of var int – интенсивность операции грузообработки (реальная).

##### Реализация:

### Ограничение на количество судов, пришвартованных у одного причала:

Будем считать, что на время операции грузообработки не требующей швартовки субъект этой операции условно пришвартован. Так же будем считать условно пришвартованным судно во время операций швартовки и отшвартовки.

Требуется чтобы количество условно либо реально пришвартованных судов у одного причала было не больше одного.

#### Константы:

**twin\_location** : array [1..n\_locations] of 1..n\_locations – локация у того же причала, отличающаяся от данной состоянием швартовки – например, локация “Причал 1, пришвартовано” будет заменена на “Причал 1, отшвартовано” и наоборот.

**related\_unmoored\_cargo\_op** : array [1..n\_locations] of set of 1..n\_operations - операции грузообработки, проводимые без пришвартовки на этой локации.

#### Переменные:

**obj\_in\_loc\_counter** : array [1..n\_locations, 0..(n\_intervals + 1)] of var int – счётчик пришвартованных судов у чётных локаций.

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % Определение obj\_in\_loc\_counter. |
| 2 |  | constraint forall (loc in 1..n\_locations, t in 0..n\_intervals where (loc mod 2) = 0) ( |
| 3 |  | obj\_in\_loc\_counter[loc, t] = (sum (obj in 1..n\_moving\_obj) (m\_obj\_loc[obj, t + 1] = loc)) % Швартовка и нахождение у причала. |
| 4 |  | + (sum (op in related\_unmoored\_cargo\_op[twin\_location[loc]]) (op\_status[op, t])) % Погрузка без швартовки. |
| 5 |  | + (sum (op in 1..n\_operations where (is\_mooring\_op[op] /\ (main\_obj\_start\_loc[op] = loc))) % Отшвартовка. |
| 6 |  | (op\_status[op, t]) |
| 7 |  | ) |
| 8 |  | ); |
| 9 |  | % Само ограничение. |
| 10 |  | constraint forall (loc in 1..n\_locations, t in 0..(n\_intervals + 1) where (loc mod 2) = 0) ( |
| 11 |  | obj\_in\_loc\_counter[loc, t] <= 1 |
| 12 |  | ); |

### Несочетаемые операции:

Из всех несочетаемых операций непокрытыми остались пары из операций грузообработки с общим субъектом, общей локацией и общим хранилищем. Эти пары отдельно вынесены в множество несочетаемых операций с ограничением, не позволяющим им выполняться вместе.

Так же временно в это множество включены пары {перемещение, грузообработка} с общим субъектом и локацией, так как это повышает производительность.

### Окна непогоды:

На всех интервалах непогоды просто ставим условие op\_status[op, t] = false.

### Фиксированные операции:

Некоторые исполнения операций могут быть зафиксированы, требуется их присутствие в ответе в том же виде. Решается обычным фиксированием нужных переменных из op\_status и participation\_as\_resource.

#### Константы:

**n\_fixed\_op** : int – количество фиксированных операций.  
**fixed\_op** : array [1..n\_fixed\_op] of 1..n\_operations – номера фиксированных операций.

**fixed\_op\_resources** : array [1..n\_fixed\_op] of set of 1..n\_moving\_obj – использованные ресурсы.

**fixed\_op\_start** : array [1..n\_fixed\_op] of 1..n\_intervals – первый интервал фиксированной операции.

**fixed\_op\_end** : array [1..n\_fixed\_op] of 1..n\_intervals – последний интервал фиксированной операции.

**is\_fixed** : array [1..n\_operations, 1..n\_intervals] of bool – фиксирована ли операция в данный момент.

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | constraint forall (no in 1..n\_fixed\_op, op = fixed\_op[no]) ( |
| 2 |  | forall (t in fixed\_op\_start[no]..fixed\_op\_end[no]) ( |
| 3 |  | (op\_status[op, t]) /\ |
| 4 |  | forall (obj in fixed\_op\_resources[no]) ( |
| 5 |  | participation\_as\_resource[obj, t] = op |
| 6 |  | ) |
| 7 |  | ) |
| 8 |  | ); |

### Критерий оптимизации:

Предыдущие ограничения описывают все логические правила, которым должно удовлетворять решение. Осталось только ввести критерий оптимальности решения – назовём решение оптимальным, если последний интервал времени, в который выполнялась хотя бы одна операция, был наименьшим среди всех возможных решений.

#### Переменные:

**is\_not\_terminated** : array [1..(n\_intervals + 1)] of var bool – означает не закончились ли все операции до текущего интервала.

#### Реализация:

|  |  |  |
| --- | --- | --- |
| 1 |  | % В конце всё остановится. |
| 2 |  | constraint (is\_not\_terminated[n\_intervals + 1] == false); |
| 3 |  | % Действия ещё не остановились, если в текущий интервал что-нибудь происходит или они не остановятся в следующий интервал. |
| 4 |  | constraint forall (t in 1..n\_intervals) ( |
| 5 |  | is\_not\_terminated[t] == ( |
| 6 |  | (exists (op in 1..n\_operations) (op\_status[op, t])) |
| 7 |  | \/ |
| 8 |  | is\_not\_terminated[t + 1] |
| 9 |  | ) |
| 10 |  | ); |
| 11 |  | solve minimize sum(is\_not\_terminated); |

### Оптимизация “сдвиг в начало” (только идея):

Основная идея – если операция может быть сдвинута на один интервал к началу без изменения других операций, то она должна быть сдвинута.

Рассмотрим операцию длительностью в один интервал. Очевидно, что она может быть выполнена в данный интервал, если:

1. Не мешают окна непогоды.
2. Главный объект на месте и не участвует в операции перемещения.
3. Достаточно свободных ресурсов на нужном месте.
4. Если это операция перемещения, то субъект не участвует в операциях погрузки.
5. Если это операция пришвартовки, то причал должен быть свободен.
6. Если это операция грузообработки, то она не выведет за границы дозволенного (меньше нуля или выше максимального объёма).
7. Если это операция грузообработки **без швартовки**, то причал должен быть свободен.
8. Не выполняется ни одной конфликтующей операции.

И сама оптимизация – если операция не фиксирована, то её невозможно сдвинуть на один интервал (в предыдущем (от старта) интервале невозможно выполнить эту операцию, считая, что её длительность равна одному кванту времени).

### Оптимизация “отсечение лишних передвижений”:

Хочется, чтобы объект не ездил с одной локации на другую, если при этом эти передвижения можно убрать без вреда для остальных операций и ограничений задачи.

Один из вариантов запретить часть перемещений – потребовать, чтобы если объект перемещается в локацию, из которой он прибыл, то он сделал или делает что-то потенциально “полезное”.

Для удобства примем необременительное предположение, что **при перемещениях и швартовках/отшвартовках ресурсами будут только буксиры**, а также что **буксиры способны дойти куда-либо своим ходом не медленнее, чем участвуя в буксировке**.

Определим полезность прибытия и отбытия объекта назад, в исходную локацию, как если выполнено хотя бы одно из условий:

* Операция прибытия, или операция отбытия, либо одна из операций между ними, в которой участвовал объект – фиксирована. (Объект выполнял одно из условий задачи.)
* Объект участвовал в качестве ресурса в одной из операции от отбытия до прибытия включительно. (Например, это буксировщик или оборудование.)
* Объект участвовал в операции грузообработки на рассматриваемом интервале (Эта операция могла изменить состояние хранилищ).
* Объект отшвартовался, выждал некоторое время и пришвартовался. (В то время, пока объект выжидал, кто-нибудь мог совершать у причала операции, которые блокировал данный объект.)

Можно показать, что во всех остальных случаях такие перемещения можно отменить без вреда для задачи.

### Оптимизация 3:

Здесь снова понадобится предположение – **транспортные корабли и бункеровщики никогда не участвуют где-либо в качестве ресурсов**. (Заметим, что при бункеровке ни сам корабль, ни бункеровщик не считаются ресурсами.)

Данная оптимизация применима только для транспортных кораблей и бункеровщиков.

Сама оптимизация – если объект в принципе не может совершить ни одной операции грузообработки в этой локации без выхода своего хранилища за дозволенные границы и объект может покинуть локацию **только** своим ходом, то объект либо уходит **немедленно**, либо остаётся в данной локации навсегда.

Другими словами, если с текущей наполненностью грузов судно никогда не сможет поучаствовать в погрузке/разгрузке в этой локации, и в то же время нет смысла дожидаться ресурсов чтобы покинуть локацию быстрее, чем самостоятельно; а так же нет смысла оставаться чтобы поучаствовать в фиксированной операции, то ничто не может помешать отправляться немедленно. Так что исключим бесполезное ожидание.