Scheduling Puzzle Game

# Introduction

The code associated with this document is a game that provides a fairly complicated scheduling environment where different scheduling algorithms and heuristics can be tested. The major components are tasks, resources and slots, with the objective being to maximize your score by placing tasks in slots on the resources.

The tasks and resources are defined in a json file. Both scoring criteria and any meta-information or requirements for the specific puzzle will be provided as a text file. Descriptions of each of the components in the json file will include the purpose of the object and the attributes of the object.

The game is played on a playing field where the size is specified in the meta-information.

## TASK

A task has a location within the playing field. It has

* ID
* Location
* Importance
* threshold
* span parameters
* distance parameters
* characteristics.

### ID

The ID uniquely identifies a task. Tasks are generally the letter ‘T’ followed by a number.

### Location

The location of a task is either a fixed point on the playing field denoted by [ x, y ], or a dynamic point where either the x or y or both can be denoted by equation in terms of **s** (which represents the slot number). For example: [ ‘pi + sin( **s** \* pi / 50 )’, ‘1 + cos( **s** \* pi / 50) / 3’ ] where

x = pi + sin( **s** \* pi / 50 )

y = 1 + cos( **s** \* pi / 50 ) / 3

Pre-defined terms include +, -, \*, /, % (modulo), \*\* (power), pi, sin, cos, tan, sqrt, and pwl

### Importance

The importance of a task is used in some games to determine which task takes priority over another task. Importance starts at one and doubles for each increment of importance. Many items can have the same importance.

### Threshold

When using slot metrics, the threshold indicates which slots can be used. This will be explained in more detail later.

### Span Parameters

The span parameters indicate how many slots a task should span. The parameters indicate a minimum span, a nominal span, and a maximum span. These parameters can be used in many ways to determine the Value associated with placing a task on a given resource. A typical interpretation would be that the task gets 70%, 85% or 100% of the value for being placed on a resource across min, nom, max slots.

### Distance Parameters

The distance parameters use the distance between the position of the resource and the location of the task to compute a distance metric using the min, nom, max paradigm. This will be explained in more detail later.

### Characteristics

The characteristics are a set of tags on the task that must be acceptable on the resource.

## RESOURCE

A resource has a position within the playing field. It has an ID, position, group, and derating

### ID

The ID uniquely identifies a resource. Resources are generally the letter ‘R’ followed by a number.

### Position

The Position of a resource is either a fixed point on the playing field denoted by [ x, y ], or a dynamic point where either the x or y or both can be denoted by equation in terms of ‘**s**’ (which represents the slot number). For example: [ ‘pi + sin( **s** \* pi / 50 )’, ‘1 + cos( **s** \* pi / 50) / 3’ ] where

x = pi + sin( **s** \* pi / 50 )

y = 1 + cos( **s** \* pi / 50 ) / 3

### Group

Resources within a group will have similar derating and characteristics.

### Derating

Is used to adjust distance metrics for specific resources within a group.

## SLOT

A slot is the basic unit for scheduling. It can represent a time unit for temporal scheduling or a bin for physical scheduling, or any other metaphor that is applicable. For now, let’s think of it as a unit of time – which could be femtoseconds or eons, it doesn’t matter. The important thing to remember is that resources have slots which are filled by tasks. More than one task can potentially fill a slot when permitted within the game.

# Example for use in this Report

Here is a small example to demonstrate these concepts. There are 50 slots. Tasks are red and resources are orange.

### The tasks: Task ID, Location x, Location y

|  |  |  |
| --- | --- | --- |
| T0 | 2 | 1 |
| T1 | 2 | 1 |
| T2 | 4 | 2 |
| T3 | 2 | -2 |
| T4 | 2 | 2 |
| T5 | 3 | 3 |
| T6 | 3 | 0 |
| T7 | 1 | 2 |
| T8 | 1 | 1 |
| T9 | 4 | -1 |
| T10 | 3 | 2 |
| T11 | 3 | -1 |
| T12 | 5 | 3 |
| T13 | 5 | 1 |
| T14 | 1 | 0 |
| T15 | 2 | -1 |

The Resources: Resource ID, Position x, Position y

|  |  |  |
| --- | --- | --- |
| R1 | 3.5 | -0.5 |
| R2 | 3.0 | -1 + 4 \* sin( (pi/100) \* s) |
| R3 | 2.5 + 2 \* cos( (pi/20) \* s ) | 3 \* sin( (pi/100) \* s ) |
| R4 | 1 + 3 \* cos( (pi/100) \* s ) | -2 + 4 \* sin( (pi/50) \* s ) |

## Json File

The next page shows an example json file for the example above.

{

"Resource" : {

"R1" : [ [ 3.5, -0.5 ], 111, 1.0 ],

"R2" : [ [ "3.0", " -1 + 4 \* sin( (pi/100) \* s)" ], 122, 1.0 ],

"R3" : [ [ "2.5 + 2 \* cos( (pi/20) \* s )", "3 \* sin( (pi/100) \* s ) " ], 133 , 0.9 ],

"R4" : [ [ "1 + 3 \* cos( (pi/100) \* s )", "-2 + 4 \* sin( (pi/50) \* s )" ], 146, 0.8 ]

},

"Task" : {

"T0" : [ [ 2, 1 ], 8, [ 7, 12, 15 ], 0.80, [ 0.4, 2.0, 3.0 ], [ 8, 61 ] ],

"T1" : [ [ 2, 1 ], 8, [ 7, 12, 15 ], 0.80, [ 0.1, 2.0, 3.0 ], [ 9, 61 ] ],

"T2" : [ [ 4, 2 ], 8, [ 9, 12, 14 ], 0.90, [ 0.1, 2.0, 3.0 ], [ 9, 61 ] ],

"T3" : [ [ 2, -2 ], 4, [ 6, 10, 12 ], 0.80, [ 0.2, 2.0, 3.0 ], [ 8, 61 ] ],

"T4" : [ [ 2, 2 ], 4, [ 5, 11, 15 ], 0.75, [ 0.1, 3.0, 5.0 ], [ 2, 61 ] ],

"T5" : [ [ 3, 3 ], 4, [ 7, 12, 18 ], 0.70, [ 0.1, 2.0, 3.0 ], [ 8, 61 ] ],

"T6" : [ [ 3, 0 ], 4, [ 5, 8, 10 ], 0.87, [ 0.2, 2.0, 3.0 ], [ 8, 61 ] ],

"T7" : [ [ 1, 2 ], 2, [ 9, 15, 20 ], 0.80, [ 0.1, 2.0, 3.0 ], [ 8, 61 ] ],

"T8" : [ [ 1, 1 ], 2, [ 7, 12, 16 ], 0.85, [ 0.3, 1.0, 3.0 ], [ 10, 61 ] ],

"T9" : [ [ 4, -1 ], 2, [ 7, 13, 16 ], 0.83, [ 0.2, 2.0, 3.0 ], [ 8, 61 ] ],

"T10" : [ [ 3, 2 ], 2, [ 9, 25, 40 ], 0.80, [ 0.4, 2.0, 3.0 ], [ 9, 61 ] ],

"T11" : [ [ 3, -1 ], 1, [ 2, 7, 9 ], 0.80, [ 0.1, 1.0, 2.0 ], [ 13, 61 ] ],

"T12" : [ [ 5, 3 ], 1, [ 2, 12, 14 ], 0.80, [ 0.2, 1.0, 2.0 ], [ 13, 61 ] ],

"T13" : [ [ 5, 1 ], 1, [ 2, 12, 14 ], 0.70, [ 0.1, 1.0, 3.0 ], [ 10, 61 ] ],

"T14" : [ [ 1, 0 ], 1, [ 2, 3, 4 ], 0.70, [ 0.1, 3.0, 5.0 ], [ 3, 61 ] ],

"T15" : [ [ 2, -1 ], 1, [ 2, 3, 4 ], 0.70, [ 0.1, 3.0, 5.0 ], [ 3, 61 ] ]

}

}

# Distance Metrics

The distance metrics reflect how “close” the task is to the resource for any slot. For each slot, there is a distance between the task and the resource on the playing field. That distance is used to compute a distance metric for each slot by applying a piecewise-linear function to assign a value between 0.0 and 1.0 to each slot. This is done for every task that is compatible with a resource by computing the distance between the task and resource for a given slot, then applying that distance as the entry to a PWL function to determine the distance metric of that slot. The PWL function is normally specified by the distance attribute of the task as a [minimum, nominal, maximum] and reflects the desired distance between the task and resource. More on this later.

Before computing the distance metrics for a task-resource pair, the characteristics of both the task and resource are compared to see if they are compatible. The characteristics of the resource are determined by the group id (more to follow). A task is compatible with a resource if every tag in the task’s characteristic list is included in the characteristics of the resource. If a task and resource are not compatible, then no metrics are computed for that pair.

The distance metrics are the most important characteristic for optimizing the schedule. When the distance metric in a slot is less than the threshold, that slot cannot be used by the task on that resource.

Using the example, if there are 50 slots and we want to place task T2 on the moving resource R3, the distance metrics would be:

[place holder]

# Groups

Each resource within a group will have similar constraints and resources. Groups are defined by a group value (100, 200, 300 or 400) and are added to a sub-group specifier (10, 20, 30, 40, and 50). Thus, the members of group 100 are R110, R120, R130, R140 and R150. The members of group 200 are R210, R220, R230, and R240. Groups 300 and 400 are similar.

Added to the group id is the specific resource id. For group 10:

* 110, 111, 112, 113, 114, 115, 116
* 120, 121, 122, 123, 124, 125, 126
* 130, 131, 132, 133, 134, 135
* 140, 141, 142, 143, 144, 145, 146
* 150, 151, 152, 153, 154, 155, 156

Within group 100 there are 7 (0-6) unique characteristic sets where the last digit of the group id specifies the characteristic set.

* 1[1-5] 0: [ 5, 61, 62 ]
* 1[1-5] 1: [ 6, 7, 8, 9, 10, 61, 62 ]
* 1[1-5] 2: [ 7, 8, 9, 61, 62 ]
* 1[1-5] 3: [ 9, 10, 11, 61, 62, 65, 66 ]
* 1[1-5] 4: [ 6, 7, 8, 61, 62 ]
* 1[1-5] 5: [ 8, 9, 10, 11, 12, 61, 62 ]
* 1[1-5] 6: [ 13, 61, 62 ]

# Meta-Information

There are many requirements, restrictions and constraints that **might** come into play. These will be described in the meta-information for the puzzle if they are required. The following are some examples of what to expect.

## Puzzle information

The size of the playing field and number of slots will be given as part of the meta-information.

## Distance constraints

When the user is allowed to place the resources on the plane, the resources within a group have distance constraints that restrict placement of that group’s resources at certain positions based on the characteristic set. The following constraints are identical across all of group 100.

* 1n0: no restriction
* 1n1 – 1n0: distance must be between 1 and 5
* 1n1 – 1n2: distance must be greater than 1
* 1n1 – 1n4: distance must be greater than 1
* 1n2 – 1n0: distance must be between 1 and 7
* 1n2 – 1n1 : distance must be greater than 1
* 1n2 – 1n4: distance must be greater than 3
* 1n3 – 1n0: distance must be between 1 and 9
* 1n3 – 1n1: distance must be greater than 2
* 1n3 – 1n2: distance must be greater than 2
* 1n3 – 1n4: distance must be greater than 2
* 1n4 – 1n0: distance must be between 2 and 11
* 1n4 – 1n1: distance must be greater than 1
* 1n4 – 1n2: distance must be greater than 3
* 1n5: no restriction
* 1n6: no restriction

## Task Pairing Requirements

In some cases multiple tasks may have a constraint that forces certain behaviors.

* Pairing – identical tasks must be placed in the same slots on two or more resources
* Abut – one task must immediately follow another on the same resource
* Overlap – one task must overlap another by a specified number of slots
* Simul – a task must be paired with an identical task on a resource in another specified group

## Task restrictions

* Annulus – a task must be within a certain range of distance from the resource
  + Example: must be between 5 and 7 distance from the resource
* Arc - a task must be within a certain arc from the resource
  + Example: must be between 30 and 75 degrees from the resource.
* Bff – a task is either recommended or required to be on a specific resource (or set of resources)
* Nogo – a task cannot be placed on the specified resource (or set of resources)

## Schedule Restrictions

Sometimes there may be broken resources (even though they are available in the json file). If that occurs, they will be noted in the meta-information with the resource and span.

# Ongoing Improvements (RFCs)

Only very specific items can be changed by the user. In the beginning most parameters and values are fixed, but as the program evolves, it will be possible for the user to:

* Specify a static position for a resource
* Specify a dynamic position for a resource
* Specify a PWL position (static positions that change at specific slots) for a resource
* Specify a position with a pair of equations (using available math functions)
* Update the importance of a task (very limited)
* Update the importance of a task with a dynamic function based on s (very limited)

## Other enhancements

Implement a very simple algorithm (plus some simple heuristics) to provide a constructive initial placement.

* ISM – (Importance, Span, Metrics): The tasks are stepped through in order of importance. The metrics are used to determine better opportunities. The task is placed on the opportunity with the highest value based on importance, span and metrics.

Use weighted opportunities when considering a task on a resource.

1. Look at each set of available slots that exceeds the minimum span
2. Remove any slots that are already scheduled with an item of higher priority
3. Using the minimum span, find the best locations – keep track of the 10 best
4. Increase the span in available slots from minimum to nominal to maximum and see if you get a better value
5. If there is an adjoining task that is greater than minimum span, see if shortening it and lengthening the current task increases the value
6. Loop back to step 4 until the maximum span is reached or there are no available slots

Allow tasks to update the resource position during the puzzle. For example if R4 is at 2, 4 to support a task in slots 0-8, it will be possible to move it to 3, 5 to support a different task in slots 12-22.

* At the beginning of the mapping, R4 is at the static location [ 5, 2 ]
* To support the two tasks, a PWL function is used
  + [ ‘pwl( s, [ [0,2], [8, 2], [12, 3] ] )’, ‘pwl( s, [ [0, 4], [8, 4], [12, 5] ] )’ ]
* Note that resource starts at 2 in slot 0, and stays at [3, 5] from 12 onward

The minimum duration for the resource to move is 1 slot. In this case we used 4 slots. If the second task needs to be at 3, 5 as soon as possible after the first task finishes we would use this function

* [ ‘pwl( s, [ [8, 2], [9, 3] ] )’, ‘pwl( s, [ [8, 4], [9, 5] ] )’ ]
* Note that the resource starts at 2, 4 (by definition of how PWL works).
* Note that the resource stays at 3, 5 from 9 onward
* More inflection points could be added for reassurance, but are not necessary

# Scoring

When a mapping is submitted, it will be scored – an overall value for the mapping will be computed based on a scoring program. The method used by the scoring program will be described in the meta-information. The scoring program will change with new puzzles.

[place holder]