**The conflict-based search algorithm (CBS)**

CBS solves MAPF by decomposing it into a large number of constrained single-agent pathfinding problems. Each of these problems can be solved in time proportional to the size of the map and length of the solution, but there may be an exponential number of such single-agent problems.

### **Definitions for CBS**

The following definitions are used in the remainder of the paper.

* We use the term *path* only in the context of a single agent and use the term *solution* to denote a set of *k* paths for the given set of *k* agents.
* A *constraint* is a tuple (*ai, v, t*) where agent *ai* is prohibited from occupying vertex *v* at time step *t*. During the course of the algorithm, agents will be associated with constraints. A *consistent path* for agent *ai* is a path that satisfies all its constraints. Likewise, a *consistent solution* is a solution that is made up from paths, such that the path for any agent *ai* is consistent with the constraints of *ai.*
* A *conflict* is a tuple (*ai, aj,, v, t*)where agent *ai* and agent *aj* occupy vertex *v* at time point *t*. A solution (of *k* paths) is *valid* if all its paths have no conflicts. A consistent solution can be *invalid* if, despite the fact that the individual paths are consistent with the constraints associated with their agents, these paths still have conflicts.

The key idea of CBS is to grow a set of constraints and find paths that are consistent with these constraints. If these paths have conflicts, and are thus invalid, the conflicts are resolved by adding new constraints.

#### **The constraint tree**

At the high level, CBS searches a tree called the constraint tree (CT). A CT is a binary tree. Each node N in the CT consists of:

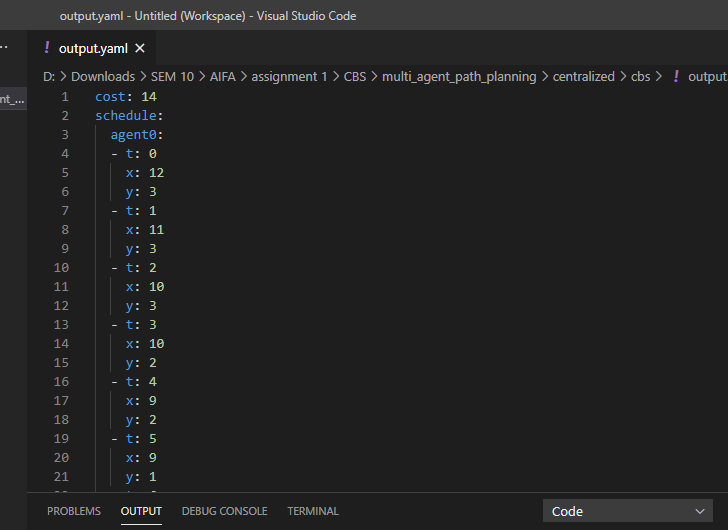
* **A set of constraints** (*N.constraints*). Each of these constraints belongs to a single agent. The root of the CT contains an empty set of constraints. The child of a node in the CT inherits the constraints of the parent and adds one new constraint for one agent.
* **A solution** (*N.solution*). A set of k paths, one path for each agent. The path for agent *ai*must be consistent with the constraints of *ai*. Such paths are found by the low-level search.
* **The total cost (***N.cost***)** of the current solution (summed over all the single-agent path costs). This cost is referred to as the f-value of node N.

Node N in the CT is a goal node when *N.Solution* is valid, i.e., the set of paths for all agents has no conflicts. The high level performs a best-first search on the CT where nodes are ordered by their costs. In our implementation, ties are broken in favour of CT nodes whose associated solution contains fewer conflicts. Further ties were broken in a FIFO manner.

**Heuristic of CBS, CBSH**:

The high level of CBS always chooses to expand the CT node N with the smallest N.cost. CBSH speeds up the high-level search through the addition of an admissible heuristic. The idea is simple: If N.solution contains one cardinal conflict, then an h-value of 1 is admissible for N because the cost of any of its descendant CT nodes with a conflict-free solution is at least N.cost + 1. If N.solution contains multiple cardinal conflicts, then CBSH builds a conflict graph, whose vertices represent agents and edges represent cardinal conflicts in N.solution. The cost of the path of at least one agent from each cardinal conflict has to increase by at least 1. Thus, the size of a minimum vertex cover (MVC) of the conflict graph (i.e., a set of vertices such that each edge is incident on at least one vertex in the set) is an admissible h-value for N. We refer to this heuristic as the CG heuristic.

**Results**:



The output is obtained in the format of an output.yaml file which tells the path traced by each robot at every time interval. Here the cost is the sum total of time taken by all the robots to reach their respective destinations.