```
import networkx as nx
import matplotlib.pyplot as plt
from swarms import Agent
from typing import List, Optional, Callable
from swarms.structs.base_swarm import BaseSwarm


class AStarSwarm(BaseSwarm):
    def __init__(
        self,
        root_agent: Agent,
        child_agents: Optional[List[Agent]] = None,
        heuristic: Optional[Callable[[Agent], float]] = None,
        *args,
        **kwargs,
    ):
        """
        Initializes the A* Swarm with a root agent and optionally a list of child agents.

        Args:
            root_agent (Agent): The root agent in the swarm.
            child_agents (Optional[List[Agent]]): List of child agents.
        """
        self.root_agent = root_agent
        self.child_agents = child_agents
        self.heuristic = heuristic
```

```
        self.child_agents = (
            child_agents if child_agents is not None else []
        )
        self.parent_map = {
            agent: root_agent for agent in self.child_agents
        }

    def a_star_communicate(
        self,
        agent: Agent,
        task: str,
    ) -> str:
        """
        Distributes the task among agents using A* search-like communication.

        Args:
            agent (Agent): The agent to start the communication from.
            task (str): The task to distribute and process.
            heuristic (Callable[[Agent], float], optional): Function to prioritize which agent to communicate
with first.

        Returns:
            str: The result of the task after processing.
        """
        # Perform the task at the current agent
        result = agent.run(task)
```

```
    # Base case: if no child agents, return the result
    if agent not in self.parent_map.values():
        return result

    # Gather child agents
    children = [
        child
        for child, parent in self.parent_map.items()
        if parent == agent
    ]

    # Sort children based on the heuristic (if provided)
    if self.heuristic:
        children.sort(key=self.heuristic, reverse=True)

    # Communicate with child agents
    for child in children:
        sub_result = self.a_star_communicate(
            child, task, self.heuristic
        )
        result += f"\n{sub_result}"

    return result

def visualize(self):
```

```
    """
        Visualizes the communication flow between agents in the swarm using networkx and matplotlib.
    """
    graph = nx.DiGraph()

    # Add edges between the root agent and child agents
    for child in self.child_agents:
        graph.add_edge(
            self.root_agent.agent_name, child.agent_name
        )
        self._add_edges(graph, child)

    # Draw the graph
    pos = nx.spring_layout(graph)
    plt.figure(figsize=(10, 8))
    nx.draw(
        graph,
        pos,
        with_labels=True,
        node_color="lightblue",
        font_size=10,
        node_size=3000,
        font_weight="bold",
        edge_color="gray",
    )
```

```
    plt.title("Communication Flow Between Agents")
    plt.show()


def _add_edges(self, graph: nx.DiGraph, agent: Agent):
    """
    Recursively adds edges to the graph for the given agent.

    Args:
        graph (nx.DiGraph): The graph to add edges to.
        agent (Agent): The current agent.
    """
    children = [
        child
        for child, parent in self.parent_map.items()
        if parent == agent
    ]
    for child in children:
        graph.add_edge(agent.agent_name, child.agent_name)
        self._add_edges(graph, child)


def run(
    self,
    task: str,
) -> str:
    """
    Start the task from the root agent using A* communication.
```

```
        Args:
            task (str): The task to execute.
            heuristic (Callable[[Agent], float], optional): Heuristic for A* communication.

        Returns:
            str: The result of the task after processing.
        """
        return self.a_star_communicate(
            self.root_agent, task, self.heuristic
        )



# # Heuristic example (can be customized)
# def example_heuristic(agent: Agent) -> float:
#     """
#     Example heuristic that prioritizes agents based on some custom logic.

#     Args:
#         agent (Agent): The agent to evaluate.

#     Returns:
#         float: The priority score for the agent.
#     """
#         # Example heuristic: prioritize based on the length of the agent's name (as a proxy for complexity)
```

```
#     return len(agent.agent_name)



# # Set up the model as provided
# api_key = os.getenv("OPENAI_API_KEY")
# model = OpenAIChat(
#     api_key=api_key, model_name="gpt-4o-mini", temperature=0.1
# )

# # Initialize root agent
# root_agent = Agent(
#     agent_name="Financial-Analysis-Agent",
#     system_prompt=FINANCIAL_AGENT_SYS_PROMPT,
#     llm=model,
#     max_loops=2,
#     autosave=True,
#     dashboard=False,
#     verbose=True,
#     streaming_on=True,
#     dynamic_temperature_enabled=True,
#     saved_state_path="finance_agent.json",
#     user_name="swarms_corp",
#     retry_attempts=3,
#     context_length=200000,
# )
```

```
# # List of child agents
# child_agents = [
#     Agent(
#         agent_name="Child-Agent-1",
#         system_prompt=FINANCIAL_AGENT_SYS_PROMPT,
#         llm=model,
#         max_loops=2,
#         autosave=True,
#         dashboard=False,
#         verbose=True,
#         streaming_on=True,
#         dynamic_temperature_enabled=True,
#         saved_state_path="finance_agent_child_1.json",
#         user_name="swarms_corp",
#         retry_attempts=3,
#         context_length=200000,
#     ),
#     Agent(
#         agent_name="Child-Agent-2",
#         system_prompt=FINANCIAL_AGENT_SYS_PROMPT,
#         llm=model,
#         max_loops=2,
#         autosave=True,
#         dashboard=False,
#         verbose=True,
#         streaming_on=True,
```

```
#         dynamic_temperature_enabled=True,
#         saved_state_path="finance_agent_child_2.json",
#         user_name="swarms_corp",
#         retry_attempts=3,
#         context_length=200000,
#     ),
# ]


# # Create the A* swarm
# swarm = AStarSwarm(
#     root_agent=root_agent,
#     child_agents=child_agents,
#     heauristic=example_heuristic,
# )


# # Run the task with the heuristic
# result = swarm.run(
#     "What are the components of a startups stock incentive equity plan",
# )
# print(result)


# # Visualize the communication flow
# swarm.visualize()
```