JPS uses a canonical ordering to break ties between paths of equal length. Diagonal moves are taken before cardinal moves. Special rules are used to wrap the canonical ordering around obstacles (from jump points) and guarantee that all states are reached.

JPS uses a jumping policy to avoid putting many states into the open list. JPS only puts the start, the goal, and jump points into the open list. JPS continually expands states until one of these are generated.

JPS performs best-first search over the states in the open list. The search is identical to A*, just over a transformed search space from the original search graph.

Our novel decomposition of JPS allows us to create new algorithms that use these canonical ordering and other components in different ways or in different settings.

Canonical Dijkstra takes advantage of the fact that in a single-source shortest-path search every state in the state space will be visited. Therefore, the jumping policy does not hurt performance.

Canonical Dijkstra writes g-costs into the closed list while jumping, but must update g-costs if lower costs are found later (purple). It is 2.5x to 4.4x faster than Dijkstra, depending on the maps.

Canonical A* just uses the canonical ordering from JPS. It expands far more states than JPS, because it doesn't jump. But, it generates far fewer states because it doesn't scan in the wrong direction for jump points to jump to.

Bounded JPS modifies the jumping rule from JPS. Instead of jumping to the next jump point, it bounds the length of a jump to some constant distance. This prevents JPS from jumping too far on large maps.

On this map, JPS will generate all the marked states no matter where the goal is.

Weighted JPS uses a different best-first search strategy, replacing A* with weighted A*. Weighted JPS trades bounded suboptimality for faster performance.