Technical Companion to

Bidirectional Search while Ensuring Meet-In-The-Middle via Effective and Efficient-to-Compute Termination Conditions

A Self-Contained Guide for CS Undergraduates

Companion to the paper by Y. Wang, E. Weiss, B. Mu, and O. Salzman
(IJCAI 2025)

Abstract

This document explains every equation, definition, algorithm, and theorem in the MEET paper in detail, assuming only undergraduate-level familiarity with graph search (A*, Dijkstra's algorithm, BFS). No prior knowledge of bidirectional heuristic search, the meet-in-the-middle property, or termination conditions is required. Each equation is accompanied by (1) a plain-English description of what it computes, (2) a symbol-by-symbol breakdown, (3) an intuitive explanation of why the computation makes sense, and (4) concrete numerical examples where helpful.

Contents

1 Background: Shortest-Path Search

1.1 The Shortest-Path Problem

Given a weighted graph $G = (S, E, c)$, a start state s_{start}, and a goal state s_{goal}, the task is to find a **path** from s_{start} to s_{goal} whose total edge cost is minimized.

Symbol breakdown:

- S: a finite set of **states** (also called nodes or vertices).
- $E \subseteq S \times S$: a set of **edges** connecting pairs of states.
- $c : E \to \mathbb{R}^{\geq \varepsilon}$: a **cost function** that assigns a positive cost to each edge. The notation $\mathbb{R}^{\geq \varepsilon}$ means all edge costs are at least $\varepsilon > 0$ (no zero-cost or negative-cost edges).
- A **path** $\pi = \langle s_0, s_1, \ldots, s_k \rangle$ is a sequence of states where each consecutive pair is connected by an edge: $(s_i, s_{i+1}) \in E$.
- The **cost of a path**: $c(\pi) := \sum_i c(s_i, s_{i+1})$, i.e., the sum of all edge costs along the path.
- An **optimal solution** is a path from s_{start} to s_{goal} with minimal cost, denoted C^*.

Numerical Example

Consider this small graph with 5 states:

Edge	Cost
(A, B)	2
(A, C)	5
(B, D)	3
(C, D)	1
(B, E)	7
(D, E)	2

With $s_{\text{start}} = A$ and $s_{\text{goal}} = E$:

- Path $A \to B \to E$: cost $= 2 + 7 = 9$.
- Path $A \to B \to D \to E$: cost $= 2 + 3 + 2 = 7$.
- Path $A \to C \to D \to E$: cost $= 5 + 1 + 2 = 8$.

The optimal solution is $A \to B \to D \to E$ with $C^* = 7$.

1.2 Unidirectional Heuristic Search (A*)

A* is the most widely known optimal shortest-path algorithm. It searches from s_{start} toward s_{goal} by maintaining a priority queue (called OPEN) of states to explore, ordered by a priority function $f(s) = g(s) + h(s)$.

Symbol breakdown:

- $g(s)$: the **cost-to-come**—the cost of the cheapest path found so far from s_{start} to state s.
- $h(s)$: the **heuristic**—an estimate of the cost of the cheapest path from s to s_{goal}.
- $f(s) = g(s) + h(s)$: the **estimated total cost** of a complete path $s_{\text{start}} \to s \to s_{\text{goal}}$ passing through s.

A* always expands (removes from OPEN and processes) the state with the smallest f-value. When it expands s_{goal}, the search terminates with an optimal solution.

What Makes a Heuristic Admissible?

A heuristic h is **admissible** if it never overestimates the true shortest-path cost:

$$\forall\, s \in S, \quad h(s) \leq h^*(s)$$

where $h^*(s)$ is the true cheapest cost from s to s_{goal}.
Admissibility guarantees that A* finds an optimal solution. Intuitively, an admissible heuristic is "optimistic"—it always believes the remaining path will be at least as cheap as (or cheaper than) reality. This ensures A* never prematurely declares a suboptimal path to be optimal.

What Makes a Heuristic Consistent?

A heuristic h is **consistent** (also called **monotone**) if, for every edge (s, s'):

$$h(s) \leq c(s, s') + h(s')$$

This is a triangle-inequality condition. Every consistent heuristic is admissible (but not vice versa). With a consistent heuristic, once A* expands a state, it has found the optimal path to that state, so it never needs to re-expand states.
In the paper, all heuristics are assumed to be **static** and **admissible**. A static heuristic never changes its values during the search. All static heuristics are admissible (as assumed in the paper), and consistent heuristics are a strict subset of admissible ones.

Numerical Example

On the graph above with $s_{\text{start}} = A$, $s_{\text{goal}} = E$:
Suppose $h(A) = 5$, $h(B) = 4$, $h(C) = 3$, $h(D) = 2$, $h(E) = 0$.
A* expands states in order of $f(s) = g(s) + h(s)$:

1. Expand A: $g(A) = 0$, $f(A) = 0+5 = 5$. Generate B with $g(B) = 2$, $f(B) = 2+4 = 6$; and C with $g(C) = 5$, $f(C) = 5+3 = 8$.
2. Expand B ($f = 6$): Generate D with $g(D) = 5$, $f(D) = 5+2 = 7$; and E with $g(E) = 9$, $f(E) = 9+0 = 9$.
3. Expand D ($f = 7$): Update E to $g(E) = 7$, $f(E) = 7+0 = 7$.
4. Expand E ($f = 7$): Goal reached! Optimal cost $= 7$.

A* expanded 4 states (A, B, D, E), skipping C entirely because its f-value ($f(C) = 8$) was never the minimum.

1.3 Bidirectional Heuristic Search (Bi-HS)

Bidirectional heuristic search (Bi-HS) runs *two* searches simultaneously:

1. A **forward search** from s_{start} toward s_{goal}.
2. A **backward search** from s_{goal} toward s_{start}.

Each search direction $\mathcal{D} \in \{\mathcal{F}, \mathcal{B}\}$ maintains its own:

- $\textsc{Open}_{\mathcal{D}}$: a priority queue of generated-but-not-yet-expanded states.
- $\textsc{Closed}_{\mathcal{D}}$: the set of already-expanded states.
- $g_{\mathcal{D}}(s)$: the cost-to-come from that direction's origin to state s.
- $h_{\mathcal{D}}(s)$: a heuristic estimating the remaining cost from s to that direction's target.
- $f_{\mathcal{D}}(s) = g_{\mathcal{D}}(s) + h_{\mathcal{D}}(s)$: the estimated total path cost through s in direction $\mathcal{D}$.

Why Bidirectional Search Can Be Faster

The key insight behind bidirectional search is geometric. In many graphs, the number of states reachable within cost r grows exponentially with r. Unidirectional search must explore all states within cost C^* of the start, covering an "area" proportional to b^{C^*} where b is the branching factor.
Bidirectional search, ideally, only needs each direction to explore states within cost $C^*/2$, covering area proportional to $2 \cdot b^{C^*/2}$. Since $2 \cdot b^{C^*/2} \ll b^{C^*}$ for large C^*, the savings can be enormous—exponential in C^*.
However, realizing this potential requires that the two searches "meet in the middle" without over-expanding or under-expanding states.

Notation conventions used in the paper:

- Subscript $\mathcal{F}$ denotes the forward direction; subscript $\mathcal{B}$ denotes backward.
- $\mathcal{D} \in \{\mathcal{F}, \mathcal{B}\}$ denotes an arbitrary direction; $\bar{\mathcal{D}}$ denotes the opposite direction (if $\mathcal{D} = \mathcal{F}$, then $\bar{\mathcal{D}} = \mathcal{B}$, and vice versa).
- $f_{\min_{\mathcal{F}}}$: the minimum f-value among all states in $\textsc{Open}_{\mathcal{F}}$.
- $g_{\min_{\mathcal{F}}}$: the minimum g-value among all states in $\textsc{Open}_{\mathcal{F}}$.
- $f_{\min} := \min(f_{\min_{\mathcal{F}}}, f_{\min_{\mathcal{B}}})$: the global minimum f-value across both open lists.
- $g_{\min} := \min(g_{\min_{\mathcal{F}}}, g_{\min_{\mathcal{B}}})$: the global minimum g-value across both open lists.

1.4 The Challenge: When Should Bidirectional Search Stop?

In unidirectional A*, termination is straightforward: stop when you expand the goal. In bidirectional search, there is no single "goal" state—instead, the two searches must *meet*.

Finding a Solution is Not Enough

Just because the forward and backward searches share a state (i.e., some state s appears in both $\textsc{Open}_{\mathcal{F}} \cap \textsc{Open}_{\mathcal{B}}$ or in $\textsc{Closed}$ of one and $\textsc{Open}$ of the other) does *not* mean the search can stop.
The path through this "meeting point" might not be optimal. There could be a cheaper path through a different meeting point that has not been discovered yet.
The challenge is deciding when to **terminate**—when to declare that no further expansion can improve the current best solution. This is the **termination condition** (TC), and it is the central topic of this paper.

2 Key Concepts: Solutions, Intersecting States, and the Meeting State

2.1 The Current Best Solution C_{curr}

As the bidirectional search progresses, it may find paths connecting s_{start} to s_{goal}. The paper tracks:

- π_{curr}: the current-best path found so far.
- C_{curr}: the cost of π_{curr}.

At the start, $C_{\text{curr}} = \infty$ (no path found yet). Whenever a cheaper path is found, C_{curr} is updated. Upon termination, if the algorithm is correct, $C_{\text{curr}} = C^*$ (the optimal cost) and $\pi_{\text{curr}} \in \Pi^*$ (the set of optimal paths).

2.2 Intersecting States and the Meeting State

Definition 2.1 (Intersecting state)**.** A state $s \in S$ is an **intersecting state** if $s \in \text{OPEN}_{\mathcal{F}} \cap \text{OPEN}_{\mathcal{B}}$, meaning it has been generated (but not necessarily expanded) by both the forward and backward searches.

The set of all intersecting states that lie on paths with a cost equal to C_{curr} is denoted $\mathcal{I}_{\text{curr}}$. The **minimum g-value** among intersecting states is:

$$g_{\min}(s) := \min(g_{\mathcal{F}}(s),\, g_{\mathcal{B}}(s))$$

Definition 2.2 (Current-best intersecting state I_b)**.**

$$I_b := \arg\min_{s \in \mathcal{I}_{\text{curr}}} g_{\min}(s)$$

In words: among all intersecting states that lie on a path of cost C_{curr}, I_b is the one with the smallest g-value (from whichever direction has the smaller g-value for that state). I_b has the minimal g-value among such states.

When the search ends with $C_{\text{curr}} = C^*$, I_b becomes the **meeting state** s_{meet}.

Numerical Example

Suppose $s_{\text{start}} = A$, $s_{\text{goal}} = E$, and during the search:

- Forward search has found: $g_{\mathcal{F}}(D) = 5$ and $g_{\mathcal{F}}(C) = 5$.
- Backward search has found: $g_{\mathcal{B}}(D) = 2$ and $g_{\mathcal{B}}(C) = 3$.
- Both D and C are in $\text{OPEN}_{\mathcal{F}} \cap \text{OPEN}_{\mathcal{B}}$, so both are intersecting states.
- Path through D: cost $= g_{\mathcal{F}}(D) + g_{\mathcal{B}}(D) = 5 + 2 = 7$.
- Path through C: cost $= g_{\mathcal{F}}(C) + g_{\mathcal{B}}(C) = 5 + 3 = 8$.
- If $C_{\text{curr}} = 7$ (best path found so far), then $\mathcal{I}_{\text{curr}} = \{D\}$ (only D lies on a path of cost 7).
- $I_b = D$ with $g_{\min}(D) = \min(5, 2) = 2$.

3 The Meet-In-The-Middle Property (MMP)

Definition 3.1 (Meet-in-the-middle property (MMP))**.** A Bi-HS algorithm **satisfies MMP** if it only expands states s with $g_{\mathcal{D}}(s) \leq C^*/2$.

What this means in plain English: An MMP algorithm never expands a state that is "too far" from its starting point. Specifically, no state expanded in the forward direction has $g_{\mathcal{F}}(s) > C^*/2$, and no state expanded in the backward direction has $g_{\mathcal{B}}(s) > C^*/2$.

Why MMP Matters

MMP ensures that both searches stay "within their half" of the optimal path. This is the formalization of the "meet in the middle" intuition: neither search goes past the midpoint. Without MMP, a bidirectional search might degenerate: one direction might explore nearly the entire path, making the other direction redundant. In the worst case, a bidirectional search without MMP could expand *more* states than unidirectional A*!
MMP guarantees the exponential savings that motivate bidirectional search in the first place. Each direction explores at most half the depth of the solution, which can yield a quadratic reduction in states expanded (or even exponential on tree-like graphs).

Numerical Example

Consider a path $A \to B \to C \to D \to E$ with all edge costs equal to 1. Then $C^* = 4$ and $C^*/2 = 2$.
An MMP algorithm would ensure:

- Forward search expands only states with $g_{\mathcal{F}} \leq 2$: states A ($g = 0$), B ($g = 1$), C ($g = 2$).
- Backward search expands only states with $g_{\mathcal{B}} \leq 2$: states E ($g = 0$), D ($g = 1$), C ($g = 2$).
- The two searches meet at C, the midpoint.

Without MMP, the forward search might push all the way to E (expanding D with $g_{\mathcal{F}}(D) = 3 > 2$), wasting effort that duplicates the backward search.

MM-optimal: A Bi-HS algorithm is **MM-optimal** if it (i) returns an optimal solution and (ii) satisfies MMP.

4 Algorithmic Framework for MM-Optimal Bi-HS

4.1 The Generic Framework (Algorithm 1)

The paper presents a generic framework that all MM-optimal Bi-HS algorithms follow. The framework has four key operations:

1. **State priority:** How states are ordered in the priority queues.
2. **State selection for expansion:** Which state to expand next from the combined queues.
3. **State pruning:** Whether a newly generated state can be safely discarded.
4. **Termination condition (TC):** When to stop the search.

The main loop of Algorithm 1:

1. Insert s_{start} into $\textsc{Open}_{\mathcal{F}}$ and s_{goal} into $\textsc{Open}_{\mathcal{B}}$.
2. **While** both $\textsc{Open}_{\mathcal{F}}$ and $\textsc{Open}_{\mathcal{B}}$ are non-empty:
   (a) Select a state s from $\textsc{Open}_{\mathcal{F}} \cup \textsc{Open}_{\mathcal{B}}$ with the minimum priority.
   (b) If the **termination condition** holds, return the current best path.
   (c) If $s \in \textsc{Open}_{\mathcal{F}}$:
      i. Remove s from $\textsc{Open}_{\mathcal{F}}$; add s to $\textsc{Closed}_{\mathcal{F}}$.
      ii. For each neighbor s' of s:
         - If we cannot improve $g_{\mathcal{F}}(s')$, skip.
         - If s' fails the **pruning test**, skip.
         - Otherwise, insert (or update) s' in $\textsc{Open}_{\mathcal{F}}$.
   (d) If $s \in \textsc{Open}_{\mathcal{B}}$: analogous operations for the backward search.
3. Return $\emptyset$ (no solution exists).

4.2 Weak Heuristics and the MMP Challenge

Weak Heuristics Can Break MMP

A heuristic h is called **weak** if, for some state s on the optimal path, $h_{\mathcal{F}}(s)$ underestimates so badly that $g_{\mathcal{F}}(s) > h_{\mathcal{F}}(s)$ even though $g_{\mathcal{F}}(s) \leq C^*/2$.
Formally, h is weak if there exists a state s with $g_{\mathcal{F}}(s) > C^*/2$ but $f_{\mathcal{F}}(s) < C^*$. Such a state would be expanded by a standard best-first search (since $f_{\mathcal{F}}(s) < C^*$), violating

MMP.
This happens when $g_{\mathcal{F}}(s) > h_{\mathcal{F}}(s)$: the heuristic underestimates so much that the f-value remains low despite the state being far from the start.

Numerical Example

Let $C^* = 10$, $\varepsilon = 1$, and consider state s with:

- $g_{\mathcal{F}}(s) = 7$ (more than $C^*/2 = 5$—violates MMP if expanded).
- $h_{\mathcal{F}}(s) = 2$ (admissible since $h^*_{\mathcal{F}}(s) = 3$, but weak since $g_{\mathcal{F}}(s) > h_{\mathcal{F}}(s)$).
- $f_{\mathcal{F}}(s) = 7 + 2 = 9 < 10 = C^*$.

A plain best-first search would expand s (since $f(s) < C^*$), violating MMP. This is the fundamental problem that MM-type algorithms must address.

5 Meet-In-The-Middle (MM): The Baseline Algorithm

5.1 MM's Priority Function

MM uses the following priority function to order states in $\textsc{Open}_{\mathcal{F}}$ (and analogously for $\textsc{Open}_{\mathcal{B}}$):

$$\mathrm{pr}_{\mathcal{F}}(s) := \max\big(f_{\mathcal{F}}(s),\ 2g_{\mathcal{F}}(s)\big) \qquad \text{(MM-Priority)}$$

What it computes in plain English: The priority of state s is the larger of two quantities: (1) the standard A* priority $f_{\mathcal{F}}(s) = g_{\mathcal{F}}(s) + h_{\mathcal{F}}(s)$, or (2) twice the cost-to-come $2g_{\mathcal{F}}(s)$.

Symbol breakdown:

- $f_{\mathcal{F}}(s) = g_{\mathcal{F}}(s) + h_{\mathcal{F}}(s)$: the standard best-first priority. Expanding states with $f_{\mathcal{F}}(s) \le C^*$ ensures optimality.
- $2g_{\mathcal{F}}(s)$: twice the cost-to-come. Expanding states with $2g_{\mathcal{F}}(s) \le C^*$ ensures $g_{\mathcal{F}}(s) \le C^*/2$, which is exactly MMP.
- $\max(\cdot,\cdot)$: by taking the maximum, MM ensures *both* conditions hold simultaneously. A state is only expanded if both $f_{\mathcal{F}}(s) \le C^*$ (optimality) and $2g_{\mathcal{F}}(s) \le C^*$ (MMP) are satisfied.

Why max(f, 2g) Guarantees Both Optimality and MMP

A state s is expanded only when $\mathrm{pr}_{\mathcal{F}}(s) \le C^*$. Since $\mathrm{pr}_{\mathcal{F}}(s) = \max(f_{\mathcal{F}}(s), 2g_{\mathcal{F}}(s))$:

- $\mathrm{pr}_{\mathcal{F}}(s) \le C^*$ implies $f_{\mathcal{F}}(s) \le C^*$: the state could lie on an optimal path (ensures optimality).
- $\mathrm{pr}_{\mathcal{F}}(s) \le C^*$ implies $2g_{\mathcal{F}}(s) \le C^*$, so $g_{\mathcal{F}}(s) \le C^*/2$ (ensures MMP).

Without the $2g$ component, weak heuristics could allow states with $g > C^*/2$ but $f < C^*$ to be expanded, violating MMP. Without the f component, we might miss states on the optimal path that have low g but high f. The max elegantly enforces both constraints.

Numerical Example

Let $C^* = 10$. Consider three states in the forward search:

State	$g_{\mathcal{F}}$	$h_{\mathcal{F}}$	$f_{\mathcal{F}}$	$\mathrm{pr}_{\mathcal{F}}$	Expanded?
s_1	3	4	7	$\max(7,6)=7$	Yes ($7 \leq 10$)
s_2	6	2	8	$\max(8,12)=12$	No ($12 > 10$; violates MMP)
s_3	4	7	11	$\max(11,8)=11$	No ($11 > 10$; $f > C^*$)

- s_1: both f and $2g$ are ≤ 10, so it is safely expanded.
- s_2: $f = 8 \leq 10$ (optimality OK), but $2g = 12 > 10$ (MMP violated). The max catches this.
- s_3: $2g = 8 \leq 10$ (MMP OK), but $f = 11 > 10$ (not on optimal path). The max catches this too.

5.2 MM's Termination Condition

MM halts the search once:

$$C_{\text{curr}} \leq \max\big(C_{\text{pr}},\ f_{\min_{\mathcal{F}}},\ f_{\min_{\mathcal{B}}},\ g_{\min_{\mathcal{F}}} + g_{\min_{\mathcal{B}}} + \varepsilon\big) \tag{MM-TC}$$

Symbol breakdown:

- C_{curr}: the cost of the best solution found so far.
- C_{pr}: the minimum priority across both open lists. This is a lower bound on the cost of any solution that might still be found.
- $f_{\min_{\mathcal{F}}}$: the minimum f-value in $\textsc{Open}_{\mathcal{F}}$. If $C_{\text{curr}} \leq f_{\min_{\mathcal{F}}}$, no state in the forward open list can lead to a better solution.
- $f_{\min_{\mathcal{B}}}$: analogous for the backward direction.
- $g_{\min_{\mathcal{F}}} + g_{\min_{\mathcal{B}}} + \varepsilon$: a lower bound on the cost of any solution that passes through a pair of not-yet-expanded states (one from each direction). The ε accounts for at least one edge connecting them.

Why each term is a lower bound: Each of the four terms inside the max is a lower bound on the cost of any undiscovered solution. If C_{curr} is at most the maximum of all these lower bounds, then C_{curr} is at most every one of them, meaning no better solution can exist.

MM's Termination Condition is Expensive to Evaluate

To compute $g_{\min_{\mathcal{F}}}$ and $f_{\min_{\mathcal{F}}}$, MM must find the minimum g-value and minimum f-value among all states in $\textsc{Open}_{\mathcal{F}}$. Since MM sorts its queue by priority $\mathrm{pr}_{\mathcal{F}} = \max(f, 2g)$—not by g or f individually—finding these minima requires iterating over **all** states in the queue. This takes $O(|\textsc{Open}_{\mathcal{F}}| + |\textsc{Open}_{\mathcal{B}}|)$ time **per iteration**. As the search progresses, these open lists grow exponentially, making each iteration increasingly expensive.
This is a major source of MM's practical inefficiency, motivating the new MEET algorithm.

6 MEET: Meet-in-the-Middle with Early and Efficient Termination

6.1 Overview of MEET

MEET is the main contribution of the paper. It is a new Bi-HS algorithm that:

1. **Guarantees MM-optimality** (returns optimal solutions while satisfying MMP).

2. **Uses efficient termination conditions** that do not require iterating over all states in the open lists.
3. **Terminates earlier than MM** by detecting when the search can safely stop, even before MM's TC triggers.
4. **Includes state pruning** to discard states that provably cannot improve the solution.

6.2 MEET's State Priority

MEET defines the priority of a state s in direction $\mathcal{D}$ identically to MM:

$$\mathrm{pr}_{\mathcal{D}}(s) := \max\big(f_{\mathcal{D}}(s),\ 2g_{\mathcal{D}}(s)\big)$$

However, there is a crucial difference in the heuristic used. MEET uses a dynamic heuristic $\bar{h}_{\mathcal{D}}$ instead of the static heuristic $h_{\mathcal{D}}$.

6.3 MEET's Dynamic Heuristic $\bar{h}$

The key innovation of MEET is the dynamic heuristic $\bar{h}_{\mathcal{D}}(s)$, which is defined as follows:

$$f_{\mathcal{D}}(s) := g_{\mathcal{D}}(s) + \bar{h}_{\mathcal{D}}(s)$$

where $\bar{h}_{\mathcal{D}}(s)$ is initialized to $h_{\mathcal{D}}(s)$ and may be updated during the search:

- If $s \notin \textsc{Closed}_{\bar{\mathcal{D}}} \cup \{\textsc{Open}_{\bar{\mathcal{D}}}\}$ and $g_{\mathcal{D}}(s) > h_{\mathcal{D}}(s)$, then $\bar{h}_{\mathcal{D}}(s) := g_{\mathcal{D}}(s)$.
- Otherwise, if $s \in \textsc{Open}_{\bar{\mathcal{D}}}$, then $\bar{h}_{\mathcal{D}}(s) := g_{\bar{\mathcal{D}}}(s)$.
- In all other cases, $\bar{h}_{\mathcal{D}}(s) := h_{\mathcal{D}}(s)$ (the original static heuristic).

What this means in plain English: MEET "improves" the heuristic on-the-fly. When MEET has actual information about state s from the opposite direction (because the backward search has already reached s with cost $g_{\mathcal{B}}(s)$), it uses that actual cost instead of the heuristic estimate. When the static heuristic is weak ($g_{\mathcal{D}}(s) > h_{\mathcal{D}}(s)$), MEET replaces it with $g_{\mathcal{D}}(s)$ to prevent MMP violations.

Why $\bar{h}$ is an Improvement

The static heuristic $h_{\mathcal{D}}(s)$ is a fixed estimate that might be inaccurate. The dynamic heuristic $\bar{h}_{\mathcal{D}}(s)$ leverages information gathered during the search:

- If the opposite direction has already found a path to s with cost $g_{\bar{\mathcal{D}}}(s)$, this is an actual cost, not an estimate—and it is still a lower bound on $h^*_{\mathcal{D}}(s)$ because the opposite direction may not have found the optimal path to s yet.
- If $g_{\mathcal{D}}(s) > h_{\mathcal{D}}(s)$, the heuristic is "weak" for this state. Setting $\bar{h}_{\mathcal{D}}(s) = g_{\mathcal{D}}(s)$ ensures that $f_{\mathcal{D}}(s) = g_{\mathcal{D}}(s) + g_{\mathcal{D}}(s) = 2g_{\mathcal{D}}(s)$, which makes the priority function naturally enforce MMP.

The paper proves (Lemma 2) that $\bar{h}_{\mathcal{D}}(s) \leq h^*_{\mathcal{D}}(s)$ whenever MEET expands s, so $\bar{h}$ remains admissible.

Numerical Example

Suppose $C^* = 10$, $\varepsilon = 1$, and consider state s in the forward direction.
Case 1: Opposite direction has reached s.
$g_{\mathcal{F}}(s) = 3$, $h_{\mathcal{F}}(s) = 2$ (weak), $g_{\mathcal{B}}(s) = 6$.

With static h: $f_{\mathcal{F}}(s) = 3 + 2 = 5$.
With dynamic $\bar{h}$: $\bar{h}_{\mathcal{F}}(s) = g_{\mathcal{B}}(s) = 6$, so $f_{\mathcal{F}}(s) = 3 + 6 = 9$.
The dynamic heuristic gives a tighter (and still admissible) estimate, since the actual remaining cost to s_{goal} through s in the backward direction is at least 6.
Case 2: Opposite direction has not reached s, but h is weak.
$g_{\mathcal{F}}(s) = 7$, $h_{\mathcal{F}}(s) = 2$.
With static h: $f_{\mathcal{F}}(s) = 7 + 2 = 9$, $\text{pr} = \max(9, 14) = 14$.
With dynamic $\bar{h}$: $\bar{h}_{\mathcal{F}}(s) = g_{\mathcal{F}}(s) = 7$, so $f_{\mathcal{F}}(s) = 7 + 7 = 14$, $\text{pr} = \max(14, 14) = 14$.
Same priority in this case, but the elevated f-value is important for the termination conditions (TC1–TC4), which use f-values.

6.4 MEET's State Selection for Expansion

Similar to MM, MEET selects the state whose priority equals $f_{\min}$:

$$f_{\mathcal{D}}(s) = f_{\min}$$

That is, MEET expands a state s from $\textsc{Open}_{\mathcal{F}} \cup \textsc{Open}_{\mathcal{B}}$ that has the minimum f-value (using the dynamic $\bar{h}$). Note that this is *not* the same as selecting the state with the minimum priority $\text{pr}_{\mathcal{D}}(s) = \max(f_{\mathcal{D}}(s), 2g_{\mathcal{D}}(s))$: the priority function determines the ordering *within* the queue, but the state actually chosen for expansion is the one with the smallest f-value across both queues.

6.5 MEET's State Pruning

Before generating a new state s in direction $\mathcal{D} \in \{\mathcal{F}, \mathcal{B}\}$, MEET tests:

$$\min\{f_{\mathcal{D}}(s),\ g_{\mathcal{D}}(s) + g_{\bar{\mathcal{D}}}(s) \text{ if } g_{\bar{\mathcal{D}}}(s) < \infty\} > C_{\text{curr}} \qquad \text{(Pruning)}$$

What it does: If the cheapest possible complete path through s (estimated using either the heuristic or actual backward cost) exceeds the current best solution cost, then s is discarded.

Why it is safe: If $f_{\mathcal{D}}(s) > C_{\text{curr}}$, then the estimated total path cost through s already exceeds the best known solution. If $g_{\mathcal{D}}(s) + g_{\bar{\mathcal{D}}}(s) > C_{\text{curr}}$ and we know the opposite-direction cost, the actual path through s via both directions costs more than C_{curr}. In either case, expanding s cannot improve C_{curr}.

Numerical Example

Suppose $C_{\text{curr}} = 8$ and we generate state s in the forward direction:
- $g_{\mathcal{F}}(s) = 5$, $\bar{h}_{\mathcal{F}}(s) = 4$, so $f_{\mathcal{F}}(s) = 9$.
- The backward search also knows s with $g_{\mathcal{B}}(s) = 4$.
- $\min(9, 5 + 4) = \min(9, 9) = 9 > 8 = C_{\text{curr}}$.
- State s is pruned—no path through s can beat $C_{\text{curr}} = 8$.

Note: MM does **not** perform pruning. This is one reason MEET can be more efficient.

7 MEET's Termination Conditions (TC1–TC4)

MEET uses four termination conditions. The search terminates when *any one* of them holds. Each condition is a sufficient reason to conclude that $C_{\text{curr}} = C^*$.

7.1 Shared Setup and Notation

When describing TC2–TC4, the paper assumes a solution has been found ($C^* \leq C_{\text{curr}}$), and defines:

- $\mathcal{H} \in \{\mathcal{F}, \mathcal{B}\}$: the direction in which I_b has the larger g-value. Formally, $g_{\mathcal{D}}(I_b) > g_{\bar{\mathcal{D}}}(I_b)$ for $\mathcal{D} = \mathcal{H}$.
- $\text{prt}_{\mathcal{H}}(I_b)$: the parent of I_b in the $\mathcal{H}$ direction search tree.
- s: the state chosen for expansion (with $f_{\mathcal{D}}(s) = f_{\min}$).
- s': a neighbor of s generated during s's expansion.
- t: the state with the minimum f-value in the opposite direction $\bar{\mathcal{D}}$, i.e., $f_{\bar{\mathcal{D}}}(t) = f_{\min_{\bar{\mathcal{D}}}}$.

Also, $S(I_b, s)$ denotes the set of states generated after I_b was found (i.e., generated by both search directions) but before s was expanded.

An auxiliary precondition "**PC 1**" is introduced:

PC 1. $\forall\, \bar{s} \in \{S(I_b, s) \neq \emptyset\}$, the inequalities $f_{\bar{\mathcal{D}}}(\bar{s}) \geq C_{\text{curr}}$ and $g_{\mathcal{H}}(\bar{s}) > g_{\mathcal{H}}(I_b)$ hold.

In plain English: every state generated since I_b was found has an f-value at least as large as C_{curr} and is farther from I_b's "deeper" direction. When PC 1 holds, no recently generated state can improve C_{curr}. When $S(I_b, s)$ is empty, PC 1 trivially holds (there are no states to check).

7.2 TC 1: No Further Expansion Can Improve C_{curr}

$$f_{\mathcal{D}}(s) \geq C_{\text{curr}} \tag{TC 1}$$

What it says: The f-value of the state about to be expanded is at least C_{curr}.

Why this is sufficient: Since states are expanded in non-decreasing order of f-value (using the dynamic $\bar{h}$), all future expansions will have f-values $\geq f_{\mathcal{D}}(s) \geq C_{\text{curr}}$. No future expansion can find a solution cheaper than C_{curr}.

Intuition: This is the simplest and most intuitive TC—it is essentially the same condition A* uses ("stop when $f_{\min} \geq C_{\text{curr}}$"), adapted for the bidirectional setting with the improved heuristic $\bar{h}$.

Numerical Example

Suppose $C_{\text{curr}} = 7$ and the state selected for expansion has $f_{\mathcal{F}}(s) = 7$.
Since $f_{\mathcal{F}}(s) = 7 \geq 7 = C_{\text{curr}}$, TC 1 fires. All remaining states in $\text{OPEN}_{\mathcal{F}}$ have f-values ≥ 7, so no better solution can exist. The search terminates with $C^* = C_{\text{curr}} = 7$.

7.3 TC 2: Both Directions Too Far Apart

$$\begin{aligned}&\text{(i) } g_{\mathcal{D}}(I_b) \leq g_{\bar{\mathcal{D}}}(I_b), \quad \text{(ii) } g_{\mathcal{D}}(s) \leq h_{\mathcal{D}}(s), \quad \text{(iii) } g_{\mathcal{D}}(s) + g_{\bar{\mathcal{D}}}(t) + \varepsilon > C_{\text{curr}},\\ &\text{(iv) } I_b \notin \{s, t\}, \quad \text{(v) } g_{\bar{\mathcal{D}}}(t) \leq h_{\bar{\mathcal{D}}}(t).\end{aligned} \tag{TC 2}$$

What it says in plain English: The lowest-priority states in both directions are so far from each other (in terms of g-values) that even connecting them with a single edge of cost ε would exceed C_{curr}. This means no new path discovered from these frontiers can beat the current best.

Symbol-by-symbol breakdown:

- (i) $g_{\mathcal{D}}(I_b) \leq g_{\bar{\mathcal{D}}}(I_b)$: I_b's g-value in direction $\mathcal{D}$ is no larger than its g-value in the opposite direction $\bar{\mathcal{D}}$; i.e., $\mathcal{D}$ is the direction in which I_b is closer to the origin.

- (ii) $g_{\mathcal{D}}(s) \leq h_{\mathcal{D}}(s)$: the heuristic is not weak for state s (ensures $\bar{h}$ equals the static h, so the condition is meaningful).
- (iii) $g_{\mathcal{D}}(s) + g_{\bar{\mathcal{D}}}(t) + \varepsilon > C_{\text{curr}}$: the sum of the minimum g-values in both directions plus the minimum possible connecting edge cost exceeds C_{curr}.
- (iv) $I_b \notin \{s, t\}$: neither s nor t is the current-best intersecting state.
- (v) $g_{\bar{\mathcal{D}}}(t) \leq h_{\bar{\mathcal{D}}}(t)$: the heuristic is not weak for t either.

Why TC 2 is Cheaper Than MM's TC

TC 2 is similar in spirit to MM's condition $g_{\min_{\mathcal{F}}} + g_{\min_{\mathcal{B}}} + \varepsilon > C_{\text{curr}}$, but there is a crucial computational difference.
MM needs $g_{\min_{\mathcal{F}}}$ and $g_{\min_{\mathcal{B}}}$, which are the minimum g-values among *all* states in each open list. Since MM's queues are ordered by $\text{pr} = \max(f, 2g)$, not by g, finding $g_{\min}$ requires scanning the entire queue: $O(|\text{OPEN}|)$ per iteration.
MEET's TC 2 uses $g_{\mathcal{D}}(s)$ (the g-value of the state being expanded—always available) and $g_{\bar{\mathcal{D}}}(t)$ (the g-value of the minimum-f state in the opposite direction—available from the queue's top element or a simple auxiliary data structure). Both are available in $O(1)$ time.

Numerical Example

Suppose $C_{\text{curr}} = 10$, $\varepsilon = 1$, and I_b has $g_{\mathcal{F}}(I_b) = 4$, $g_{\mathcal{B}}(I_b) = 6$.
- Forward expanding s: $g_{\mathcal{F}}(s) = 5$, $h_{\mathcal{F}}(s) = 6$, $f_{\mathcal{F}}(s) = 11$.
- Backward minimum: t with $g_{\mathcal{B}}(t) = 5$, $h_{\mathcal{B}}(t) = 7$, $f_{\mathcal{B}}(t) = 12$.
- $g_{\mathcal{F}}(I_b) = 4 \leq 6 = g_{\mathcal{B}}(I_b)$: condition (i) holds ($I_b$ is closer in the forward direction).
- $g_{\mathcal{F}}(s) = 5 \leq 6 = h_{\mathcal{F}}(s)$: condition (ii) holds (heuristic is not weak).
- $g_{\mathcal{F}}(s) + g_{\mathcal{B}}(t) + \varepsilon = 5 + 5 + 1 = 11 > 10 = C_{\text{curr}}$: condition (iii) holds.
- $I_b \notin \{s, t\}$: condition (iv) holds.
- $g_{\mathcal{B}}(t) = 5 \leq 7 = h_{\mathcal{B}}(t)$: condition (v) holds.
- All conditions hold, so TC 2 fires. The search terminates with $C^* = 10$.

7.4 TC 3: No Child Can Improve After I_b (When PC 1 Fails)

TC 3 requires that PC 1 does *not* hold, and the following conditions are all satisfied:

$$\begin{aligned} &\text{(i) } g_{\mathcal{D}}(s) \geq g_{\mathcal{H}}(\text{prt}_{\mathcal{H}}(I_b)), \quad \text{(ii) } g_{\mathcal{D}}(s) > h_{\mathcal{D}}(s), \\ &\text{(iii) } c(s, s') = \varepsilon, \quad \text{(iv) } g_{\mathcal{H}}(s') > g_{\mathcal{H}}(I_b). \end{aligned} \tag{TC 3}$$

What it says in plain English: We are expanding state s and generating its child s'. TC 3 checks whether this child s' could possibly lead to a solution better than C_{curr}. If the parent of I_b (in the direction where I_b is "deeper") is no farther than s, and the newly generated child s' is farther than I_b in that direction, and the connecting edge has minimum cost ε, then no further expansions can improve C_{curr}.

The intuition: If C_{curr} is not yet optimal, there must exist some state $\bar{s}$ with $f_{\{\mathcal{D},\bar{\mathcal{D}}\}}(\bar{s}) \leq 2g_{\mathcal{D}}(I_b)$ (because any improving path must pass through a state closer to both directions than I_b). TC 3 checks that no such $\bar{s}$ has been generated—if the cheapest possible child already overshoots, no improvement is possible.

TC 3 and TC 4 Use the Minimum Edge Cost ε

Both TC 3 and TC 4 rely on $c(s, s') = \varepsilon$, where ε is the minimum edge cost in the graph. In settings where all edges have the same cost (like grid worlds with uniform movement costs), ε is simply that edge cost, and these conditions fire often.
In settings where edge costs vary widely, the condition $c(s, s') = \varepsilon$ is rarely satisfied, making TC 3 and TC 4 less effective. The paper discusses that inflating ε by a scalar w could be used to obtain bounded-suboptimal solutions, trading optimality for earlier termination.

7.5 TC 4: Similar to TC 3 with a Tighter Cost Bound

$$\begin{aligned} &\text{(i) } g_{\mathcal{D}}(I_b) \leq g_{\mathcal{D}}(s), \quad \text{(ii) } g_{\mathcal{D}}(s) > h_{\mathcal{D}}(s), \\ &\text{(iii) } c(s, s') = \varepsilon, \quad \text{(iv) } f_{\mathcal{D}}(s') \geq C_{\text{curr}}. \end{aligned} \tag{TC 4}$$

How it differs from TC 3: TC 4 replaces condition (iv) of TC 3 ($g_{\mathcal{H}}(s') > g_{\mathcal{H}}(I_b)$) with $f_{\mathcal{D}}(s') \geq C_{\text{curr}}$. This means TC 4 checks whether the newly generated child has an f-value that already meets or exceeds C_{curr}, rather than checking the g-value in a specific direction.

Also, TC 4 applies when $g_{\mathcal{D}}(I_b) \leq g_{\mathcal{D}}(s)$ (slightly different from TC 3 which checks I_b's parent).

The logic: The key differences from TC 3 are:

1. TC 4 checks $f_{\{\mathcal{D},\bar{\mathcal{D}}\}}(\bar{s}) < C_{\text{curr}}$ instead of $f < 2g_{\mathcal{D}}(I_b)$, making it a tighter check.
2. TC 4 is triggered when $g_{\mathcal{D}}(I_b) \leq g_{\mathcal{D}}(s)$.

TC 3 and TC 4 in Action

Suppose $C_{\text{curr}} = 8$, $\varepsilon = 1$, and we are expanding state s in the forward direction, generating child s':

- $g_{\mathcal{F}}(s) = 4$, $h_{\mathcal{F}}(s) = 3$ (weak: $g > h$), $f_{\mathcal{F}}(s) = 7$.
- s' is a neighbor of s with $c(s, s') = 1 = \varepsilon$.
- $g_{\mathcal{F}}(s') = 5$, $\bar{h}_{\mathcal{F}}(s') = 4$, $f_{\mathcal{F}}(s') = 9$.
- I_b has $g_{\mathcal{F}}(I_b) = 3$, $g_{\mathcal{B}}(I_b) = 5$ (so $\mathcal{H} = \mathcal{B}$ since $g_{\mathcal{B}}(I_b) > g_{\mathcal{F}}(I_b)$).
- $\text{prt}_{\mathcal{B}}(I_b)$ has $g_{\mathcal{B}} = 4$.

Check TC 4:

- (i) $g_{\mathcal{F}}(I_b) = 3 \leq 4 = g_{\mathcal{F}}(s)$. ✓
- (ii) $g_{\mathcal{F}}(s) = 4 > 3 = h_{\mathcal{F}}(s)$. ✓
- (iii) $c(s, s') = 1 = \varepsilon$. ✓
- (iv) $f_{\mathcal{F}}(s') = 9 \geq 8 = C_{\text{curr}}$. ✓

All conditions hold, so TC 4 fires. The search terminates with $C^* = C_{\text{curr}} = 8$.

7.6 Summary of All Four TCs

TC	Core Check	Intuition
TC 1	$f_{\mathcal{D}}(s) \geq C_{\text{curr}}$	The minimum f-value state already costs as much as C_{curr}. No improvement possible. Like A*'s stopping rule.
TC 2	$g_{\mathcal{D}}(s) + g_{\bar{\mathcal{D}}}(t) + \varepsilon > C_{\text{curr}}$	The two search frontiers are too far apart; any connection costs more than C_{curr}.
TC 3	Child s' has $g_{\mathcal{H}}(s') > g_{\mathcal{H}}(I_b)$ with $c(s, s') = \varepsilon$	After expanding near I_b, the cheapest child overshoots. No improving path can exist through the region around I_b.
TC 4	Child s' has $f_{\mathcal{D}}(s') \geq C_{\text{curr}}$ with $c(s, s') = \varepsilon$	The cheapest child's f-value already matches C_{curr}. No cheaper path via this child.

Why Multiple TCs Help

No single TC dominates in all situations:

- **TC 1** is the "catch-all" that always eventually fires, but it may fire late.
- **TC 2** fires early when the heuristic is strong (not weak), as it exploits the gap between the two search frontiers.
- **TC 3** and **TC 4** fire early when the minimum edge cost ε is meaningful (e.g., grid worlds), as they detect that even the cheapest child cannot improve C_{curr}.

The paper's experiments show that the distribution of which TC fires varies by domain:

- In grid maps with accurate heuristics (e.g., Euclidean distance), TC 1 dominates ($\sim$72% on brc203).
- With less accurate heuristics (e.g., Octile distance on orz100d), TC 3 and TC 4 become more frequent ($\sim$57% for TC 3, $\sim$34% for TC 4).
- In Pancake puzzles with weak heuristics (GAP-4), TC 3 dominates ($\sim$40%) alongside TC 4 ($\sim$60%).

8 Theoretical Properties and Guarantees

8.1 Lemma 1: MEET Satisfies MMP

Lemma 8.1. *MEET never expands a state s with $g_{\mathcal{D}}(s) > C^*/2$.*

What this says: MEET satisfies the meet-in-the-middle property. No state expanded in either direction has a cost-to-come exceeding half the optimal solution cost.

Proof sketch: Assume for contradiction that MEET expands s with $g_{\mathcal{D}}(s) > C^*/2$. When s is generated, MEET sets $\bar{h}_{\mathcal{D}}(s)$ to either $h_{\mathcal{D}}(s)$, $g_{\mathcal{D}}(s)$, or $g_{\bar{\mathcal{D}}}(s)$. In all cases:

$$f_{\mathcal{D}}(s) = g_{\mathcal{D}}(s) + \bar{h}_{\mathcal{D}}(s) \geq g_{\mathcal{D}}(s) + g_{\mathcal{D}}(s) = 2g_{\mathcal{D}}(s) > C^*$$

(The inequality $\bar{h}_{\mathcal{D}}(s) \geq g_{\mathcal{D}}(s)$ holds because either $\bar{h}$ was set to $g_{\mathcal{D}}(s)$ directly, or the static heuristic $h_{\mathcal{D}}(s) \geq g_{\mathcal{D}}(s)$ since the heuristic is not weak, or $g_{\bar{\mathcal{D}}}(s) \geq h^*_{\mathcal{D}}(s) \geq g_{\mathcal{D}}(s)$ in certain cases.)

Since MEET expands states in non-decreasing order of f-value and $f_{\mathcal{D}}(s) = f_{\min}$, all states with $f < C^*$ have already been expanded. But then the optimal solution would have been found and $C_{\text{curr}} = C^*$, meaning s would not be selected (because $f_{\mathcal{D}}(s) > C^* = C_{\text{curr}}$ triggers TC 1). Contradiction.

Numerical Example

Let $C^* = 10$. MEET guarantees that no expanded state has $g > 5$ in either direction. If a state s with $g_{\mathcal{F}}(s) = 6$ is generated, then:

$$f_{\mathcal{F}}(s) \geq 2 \times 6 = 12 > 10 = C^*$$

So s will never be expanded (TC 1 will fire first, since all states with $f < 10$ will have been processed by then).

8.2 Lemma 2: The Dynamic Heuristic $\bar{h}$ is Admissible at Expansion

Lemma 8.2. *When MEET expands a state s, $\bar{h}_{\mathcal{D}}(s) \leq h^*_{\mathcal{D}}(s)$.*

What this says: The dynamic heuristic never overestimates the true remaining cost at the time of expansion. This is critical because optimality proofs for best-first search require admissibility.

Proof sketch: By Lemma 1, when s is expanded, $g_{\mathcal{D}}(s) \leq C^*/2$. Since $f_{\mathcal{D}}(s) = g_{\mathcal{D}}(s) + \bar{h}_{\mathcal{D}}(s) \leq C^*$ (otherwise TC 1 would fire), we get:

$$\bar{h}_{\mathcal{D}}(s) = f_{\mathcal{D}}(s) - g_{\mathcal{D}}(s) \leq C^* - g_{\mathcal{D}}(s)$$

We also know from the definition of C^* and the admissibility of the original h that:

$$h^*_{\mathcal{D}}(s) = f^*_{\mathcal{D}}(s) - g^*_{\mathcal{D}}(s) \geq C^* - g_{\mathcal{D}}(s)$$

(since $g^*_{\mathcal{D}}(s) \leq g_{\mathcal{D}}(s)$ and $f^*_{\mathcal{D}}(s) \geq C^*$ for any state on a path through s).

More directly: substituting $g^*_{\mathcal{D}}(s) \leq g_{\mathcal{D}}(s)$ into $h^*_{\mathcal{D}}(s) = f^*_{\mathcal{D}}(s) - g^*_{\mathcal{D}}(s) \geq C^* - g_{\mathcal{D}}(s)$, we get $\bar{h}_{\mathcal{D}}(s) \leq h^*_{\mathcal{D}}(s)$.

8.3 Lemmas 3–6: Each TC Ensures Optimality

The paper proves four lemmas, one for each termination condition, all concluding that if the respective TC holds, then $C_{\text{curr}} = C^*$. We summarize the key argument for each:

Lemma 3 (TC 1 $\Rightarrow C_{\text{curr}} = C^*$): States are expanded in non-decreasing f-order (using the admissible $\bar{h}$ from Lemma 2). Once TC 1 fires ($f_{\mathcal{D}}(s) \geq C_{\text{curr}}$), all remaining states have $f \geq C_{\text{curr}}$, meaning no cheaper solution exists.

Lemma 4 (TC 2 $\Rightarrow C_{\text{curr}} = C^*$): If $g_{\mathcal{D}}(s) + g_{\bar{\mathcal{D}}}(t) + \varepsilon > C_{\text{curr}}$, then $2(g_{\mathcal{D}}(s) + \varepsilon) > C_{\text{curr}}$ and $2(g_{\bar{\mathcal{D}}}(t) + \varepsilon) > C_{\text{curr}}$ (proven by contradiction). This means any undiscovered path would have to pass through states with $g > C_{\text{curr}}/2$ in at least one direction—but $f_{\mathcal{D}}(s_i) \leq C^*$ must hold for states on the optimal path before C^* is found, so $C_{\text{curr}} > C^*$ is impossible. Hence $C_{\text{curr}} = C^*$.

Lemma 5 (TC 3 $\Rightarrow C_{\text{curr}} = C^*$): If $C_{\text{curr}} > C^*$, then expanding s after I_b must eventually generate a state $\bar{s}$ with $f_{\{\mathcal{D},\bar{\mathcal{D}}\}}(\bar{s}) < C_{\text{curr}} < 2g_{\mathcal{D}}(I_b)$. By contradiction, TC 3 requires that no such $\bar{s}$ exists among the children of s, and PC 1 failing means earlier states also could not improve. Hence $C_{\text{curr}} = C^*$.

Lemma 6 (TC 4 $\Rightarrow C_{\text{curr}} = C^*$): Similar argument to Lemma 5, using the tighter bound $f_{\mathcal{D}}(s') \geq C_{\text{curr}}$ instead of the g-value comparison.

8.4 Theorem 1: MEET Returns an Optimal Solution

Theorem 8.3. *MEET returns an optimal solution, if a solution exists.*

What this says: Combining Lemmas 1–6, MEET is **MM-optimal**:

1. It is **optimal**: when any TC fires, $C_{\text{curr}} = C^*$ (Lemmas 3–6).
2. It satisfies **MMP**: no expanded state has $g > C^*/2$ (Lemma 1).
3. It is **complete**: if a solution exists, TC 1 must eventually hold (since the search explores states in non-decreasing f-order, it will eventually reach $f \geq C^* = C_{\text{curr}}$).

9 Computational Efficiency: Why MEET is Faster Than MM

9.1 The Cost of MM's Termination Condition

MM's TC requires computing $g_{\min_{\mathcal{F}}}$, $g_{\min_{\mathcal{B}}}$, $f_{\min_{\mathcal{F}}}$, and $f_{\min_{\mathcal{B}}}$ at each iteration. Since MM's priority queue is ordered by $\text{pr} = \max(f, 2g)$—not by g or f—these minima are not "at the top of the queue." Finding them requires scanning the entire queue.

Per-iteration cost: $O(|\text{OPEN}_{\mathcal{F}}| + |\text{OPEN}_{\mathcal{B}}|)$.

As the search progresses, the open lists grow. In many practical settings, the open lists grow *exponentially* in the solution depth. This makes each iteration of MM progressively slower.

9.2 Why MEET's TCs are $O(1)$ to Evaluate

MEET's four TCs use only quantities that are immediately available:

- $f_{\mathcal{D}}(s)$: the f-value of the state being expanded (always known).
- $g_{\mathcal{D}}(s)$: the g-value of the state being expanded (always known).
- $g_{\bar{\mathcal{D}}}(t)$: the g-value of the minimum-f state in the opposite direction (maintained as a simple variable or at the top of the opposite queue).
- C_{curr}: the current best solution cost (maintained as a global variable).
- $g_{\mathcal{H}}(I_b)$, $\text{prt}_{\mathcal{H}}(I_b)$: properties of the current best intersecting state (maintained incrementally).
- $c(s, s')$: the cost of the edge just traversed (known during expansion).
- $f_{\mathcal{D}}(s')$, $g_{\mathcal{H}}(s')$: properties of the just-generated child (computed during expansion).

None of these require scanning the open lists. All are available in $O(1)$ time.

The Fundamental Speed Advantage

MM: each iteration costs $O(|\text{OPEN}|)$ to check the TC. MEET: each iteration costs $O(1)$ to check all four TCs.
On problems where $|\text{OPEN}|$ grows to millions of states, this difference is the primary reason MEET outperforms MM by two or more orders of magnitude.

9.3 Additionally: State Pruning Reduces the Search Space

MEET prunes states that provably cannot improve C_{curr}. MM does not prune any states. Pruning reduces the number of states that enter the open lists, which in turn:

1. Reduces the number of iterations (fewer states to expand).
2. Keeps the open lists smaller (benefiting queue operations even beyond the TC evaluation).

The paper reports that pruning reduces runtime by approximately 2% on average for grid maps. While modest, this compounds with the TC efficiency gains.

10 Experimental Results

10.1 Domains

The paper evaluates MEET on two domains:

Grid Maps: Two maps from the MovingAI benchmark (Dragon Age: Origins):

- *brc203d* (274×391): 1290 instances. Moderate difficulty.
- *orz100d* (412×395): 2420 instances. More challenging (heuristics are less informative).

8-way movement with Euclidean and Octile distance heuristics.

10-Pancake Puzzle: A state is a permutation of 10 numbers; the goal is to sort them via prefix reversals. Heuristics: GAP-X for $X \in \{1, 2, 3, 4\}$, where larger X means a less accurate (weaker) heuristic. 30 random instances with C^* between 7 and 10.

10.2 Key Results

Domain	Heuristic	A*	MM	BAE*	MEET
brc203	Euclidean	1.4 ms	216 ms	3.9 ms	2.2 ms
brc203	Octile	1.6 ms	406 ms	4.3 ms	2.4 ms
orz100d	Euclidean	8.5 ms	3,467 ms	23.8 ms	13.4 ms
orz100d	Octile	9.2 ms	5,214 ms	25 ms	14.7 ms
10-Pancake	GAP-1	0.31 ms	7.1 ms	0.46 ms	0.48 ms
10-Pancake	GAP-2	5.6 ms	447 ms	1.3 ms	1.0 ms
10-Pancake	GAP-3	81 ms	12,189 ms	8.2 ms	3.9 ms
10-Pancake	GAP-4	510 ms	36,133 ms	26.1 ms	6.5 ms

All values are median runtimes.

Key observations:

1. **MEET vs. MM:** MEET is at least **100× faster** than MM in all settings, and over **5,000× faster** on 10-Pancake with GAP-4. This confirms that MM's expensive TC evaluation is its primary bottleneck.
2. **MEET vs. A*:** MEET is competitive with A* on grid maps (roughly 1.5×–2× slower) and **faster than A*** on Pancake puzzles with weak heuristics (up to 78× faster with GAP-4). Bidirectional search shines precisely when heuristics are weak and the search space is large.
3. **MEET vs. BAE*:** MEET consistently outperforms BAE* (the state-of-the-art non-MM bidirectional algorithm), especially with weaker heuristics. With GAP-4, MEET is 4× faster than BAE*.
4. **The weaker the heuristic, the bigger MEET's advantage:** As the heuristic quality degrades (GAP-1 → GAP-4), MEET's advantage over all competitors grows dramatically. This is because MEET's TC 3 and TC 4 become increasingly valuable when the heuristic is weak and TC 1 alone would fire late.

Concrete Timing Comparison: 10-Pancake GAP-4

On the 10-Pancake puzzle with GAP-4 heuristic (the weakest):

- A*: 510 ms (median).

- MM: 36,133 ms (median)—**71× slower than A***! MM's expensive TC makes it impractical despite its theoretical guarantees.
- BAE*: 26.1 ms—fast, but does not guarantee MMP.
- MEET: 6.5 ms—**78× faster than A***, **5,559× faster than MM**, and **4× faster than BAE***, while guaranteeing both optimality and MMP.

11 Putting It All Together: A Complete Walkthrough

Let us trace MEET's execution on a small example to see how all components interact.

Full MEET Walkthrough

Consider a graph with states $\{A, B, C, D, E\}$, $s_{\text{start}} = A$, $s_{\text{goal}} = E$, $\varepsilon = 1$:

Edge	Cost
(A, B)	2
(A, C)	3
(B, D)	2
(C, D)	1
(D, E)	2

Optimal path: $A \to B \to D \to E$, cost $C^* = 6$. Alternative: $A \to C \to D \to E$, cost $= 6$ (also optimal).
Heuristics (admissible): $h_{\mathcal{F}}(A) = 4$, $h_{\mathcal{F}}(B) = 3$, $h_{\mathcal{F}}(C) = 2$, $h_{\mathcal{F}}(D) = 2$, $h_{\mathcal{F}}(E) = 0$. $h_{\mathcal{B}}(E) = 4$, $h_{\mathcal{B}}(D) = 3$, $h_{\mathcal{B}}(C) = 3$, $h_{\mathcal{B}}(B) = 2$, $h_{\mathcal{B}}(A) = 0$.

Initialization:

- $\textsc{Open}_{\mathcal{F}} = \{A\}$: $g_{\mathcal{F}}(A) = 0$, $\bar{h}_{\mathcal{F}}(A) = 4$, $f_{\mathcal{F}}(A) = 4$, $\text{pr} = \max(4, 0) = 4$.
- $\textsc{Open}_{\mathcal{B}} = \{E\}$: $g_{\mathcal{B}}(E) = 0$, $\bar{h}_{\mathcal{B}}(E) = 4$, $f_{\mathcal{B}}(E) = 4$, $\text{pr} = \max(4, 0) = 4$.
- $C_{\text{curr}} = \infty$.

Step 1: Expand A (forward, pr $= 4$).

- Check TCs: $C_{\text{curr}} = \infty$, so no TC fires.
- Generate B: $g_{\mathcal{F}}(B) = 2$, $\bar{h}_{\mathcal{F}}(B) = 3$, $f_{\mathcal{F}}(B) = 5$, $\text{pr} = \max(5, 4) = 5$.
- Generate C: $g_{\mathcal{F}}(C) = 3$, $\bar{h}_{\mathcal{F}}(C) = 2$, $f_{\mathcal{F}}(C) = 5$, $\text{pr} = \max(5, 6) = 6$.
- $\textsc{Open}_{\mathcal{F}} = \{B, C\}$.

Step 2: Expand E (backward, pr $= 4$).

- Check TCs: $C_{\text{curr}} = \infty$, so no TC fires.
- Generate D: $g_{\mathcal{B}}(D) = 2$, $\bar{h}_{\mathcal{B}}(D) = 3$, $f_{\mathcal{B}}(D) = 5$, $\text{pr} = \max(5, 4) = 5$.
- $\textsc{Open}_{\mathcal{B}} = \{D\}$.

Step 3: Expand B (forward, pr $= 5$) or D (backward, pr $= 5$). Say we expand B forward.

- Check TCs: $C_{\text{curr}} = \infty$, no TC fires.
- Generate D: $g_{\mathcal{F}}(D) = 4$, and $D \in \textsc{Open}_{\mathcal{B}}$ with $g_{\mathcal{B}}(D) = 2$. So $\bar{h}_{\mathcal{F}}(D) = g_{\mathcal{B}}(D) = 2$, $f_{\mathcal{F}}(D) = 6$, $\text{pr} = \max(6, 8) = 8$.
- D is now an intersecting state! Path cost through D: $g_{\mathcal{F}}(D) + g_{\mathcal{B}}(D) = 4 + 2 = 6$.
- Update $C_{\text{curr}} = 6$, $I_b = D$, $g_{\min}(D) = \min(4, 2) = 2$.

Step 4: Expand D (backward, pr $= 5$).

- Check TC 1: $f_{\mathcal{B}}(D) = 5 < 6 = C_{\text{curr}}$. TC 1 does not fire.
- Generate B backward: $g_{\mathcal{B}}(B) = 4$, $B \in \textsc{Open}_{\mathcal{F}}$ with $g_{\mathcal{F}}(B) = 2$. Path through B: $g_{\mathcal{F}}(B) + g_{\mathcal{B}}(B) = 2 + 4 = 6 = C_{\text{curr}}$ (no improvement).
- Generate C backward: $g_{\mathcal{B}}(C) = 3$, $\bar{h}_{\mathcal{B}}(C) = 3$, $f_{\mathcal{B}}(C) = 6$, $\text{pr} = \max(6, 6) = 6$.
- Pruning test for C: $\min(f_{\mathcal{B}}(C), g_{\mathcal{B}}(C) + g_{\mathcal{F}}(C)) = \min(6, 3 + 3) = 6 \leq 6 = C_{\text{curr}}$.

Not pruned (not strictly greater).

Step 5: Next expansion. $\textsc{Open}_{\mathcal{F}} = \{C, D\}$ with $\text{pr}(C) = 6$, $\text{pr}(D) = 8$. $\textsc{Open}_{\mathcal{B}} = \{B, C\}$ with $\text{pr}(B) = \max(6, 8) = 8$, $\text{pr}(C) = 6$.

Minimum priority is 6, so we expand C (from either direction).

- Check TC 1: $f(C) = 6 \geq 6 = C_{\text{curr}}$. **TC 1 fires!**
- Search terminates. $C^* = C_{\text{curr}} = 6$.
- MMP check: all expanded states had $g \leq 2 \leq C^*/2 = 3$? Let us verify: $g_{\mathcal{F}}(A) = 0$, $g_{\mathcal{F}}(B) = 2$, $g_{\mathcal{B}}(E) = 0$, $g_{\mathcal{B}}(D) = 2$. All ≤ 3. ✓

The search expanded only 4 states (A, B, E, D), meeting at D with g-values $\leq C^*/2 = 3$ in both directions.

12 Summary of Notation

Symbol	Type	Meaning
$G = (S, E, c)$	Graph	Weighted graph with states S, edges E, costs c
$s_{\text{start}}, s_{\text{goal}}$	States	Start and goal states
ε	Scalar > 0	Minimum edge cost in G
π	Path	Sequence of states $\langle s_0, \ldots, s_k \rangle$
$c(\pi)$	Scalar ≥ 0	Cost of path π (sum of edge costs)
C^*	Scalar ≥ 0	Optimal solution cost
Π^*	Set of paths	Set of all optimal solutions
C_{curr}	Scalar ≥ 0	Cost of the best solution found so far
π_{curr}	Path	Current best solution path
$\mathcal{F}$	Direction	Forward search direction ($s_{\text{start}} \to s_{\text{goal}}$)
$\mathcal{B}$	Direction	Backward search direction ($s_{\text{goal}} \to s_{\text{start}}$)
$\mathcal{D}$	Direction	Arbitrary direction $\in \{\mathcal{F}, \mathcal{B}\}$
$\bar{\mathcal{D}}$	Direction	Opposite of $\mathcal{D}$
$g_{\mathcal{D}}(s)$	Scalar ≥ 0	Cost-to-come: cheapest known path from origin of $\mathcal{D}$ to s
$h_{\mathcal{D}}(s)$	Scalar ≥ 0	Static heuristic: estimate of remaining cost from s
$h^*_{\mathcal{D}}(s)$	Scalar ≥ 0	True optimal remaining cost from s
$\bar{h}_{\mathcal{D}}(s)$	Scalar ≥ 0	MEET's dynamic heuristic (improved h)
$f_{\mathcal{D}}(s)$	Scalar ≥ 0	$g_{\mathcal{D}}(s) + \bar{h}_{\mathcal{D}}(s)$: estimated total path cost through s
$\text{pr}_{\mathcal{D}}(s)$	Scalar ≥ 0	$\max(f_{\mathcal{D}}(s), 2g_{\mathcal{D}}(s))$: priority in queue
$\textsc{Open}_{\mathcal{D}}$	Priority queue	Generated but not-yet-expanded states in direction $\mathcal{D}$
$\textsc{Closed}_{\mathcal{D}}$	Set	Expanded states in direction $\mathcal{D}$
$f_{\text{min}_{\mathcal{D}}}$	Scalar	Minimum f-value in $\textsc{Open}_{\mathcal{D}}$
$g_{\text{min}_{\mathcal{D}}}$	Scalar	Minimum g-value in $\textsc{Open}_{\mathcal{D}}$
f_{min}	Scalar	$\min(f_{\text{min}_{\mathcal{F}}}, f_{\text{min}_{\mathcal{B}}})$
g_{min}	Scalar	$\min(g_{\text{min}_{\mathcal{F}}}, g_{\text{min}_{\mathcal{B}}})$
$\text{nbr}(s)$	Set	Neighbors of s: $\{s' \mid (s, s') \in E\}$
$\text{prt}(s)$	State	Parent of s in the search tree
$g_{\text{min}}(s)$	Scalar	$\min(g_{\mathcal{F}}(s), g_{\mathcal{B}}(s))$
$\mathcal{I}_{\text{curr}}$	Set	Intersecting states on paths of cost C_{curr}
I_b	State	Current-best intersecting state ($\arg\min_{s \in \mathcal{I}_{\text{curr}}} g_{\text{min}}(s)$)
s_{meet}	State	Meeting state ($= I_b$ when $C_{\text{curr}} = C^*$)
$\mathcal{H}$	Direction	Direction where I_b has larger g-value
$S(I_b, s)$	Set	States generated after I_b found, before s expanded
TC 1–TC 4	Conditions	MEET's four termination conditions
PC 1	Precondition	Auxiliary condition for TC 3 and TC 4
MMP	Property	Meet-in-the-middle property: $g_{\mathcal{D}}(s) \leq C^*/2$ for all expanded s

13 Frequently Asked Questions

Q: Why not just run A* if it is already fast on grid maps?

A: On grid maps with accurate heuristics, A* is indeed competitive (and sometimes faster than MEET due to lower overhead). However, A*'s performance degrades sharply as heuristic quality decreases. On the 10-Pancake puzzle with GAP-4, A* takes 510 ms while MEET takes 6.5 ms—nearly 80× faster. Bidirectional search is most valuable precisely when heuristics are weak, because it effectively "searches from both ends" to compensate for poor cost estimates.

Q: Why is MM so much slower than MEET if they have the same priority function?

A: The priority function is the same, but the termination condition is fundamentally different. MM must scan both entire open lists every iteration to compute $g_{\min}$ and $f_{\min}$ (an $O(|\textsc{Open}|)$ operation). MEET's TCs use only locally available quantities ($O(1)$ per iteration). As the open lists grow to hundreds of thousands or millions of states, MM spends most of its time just checking whether to stop, not actually searching.

Q: Does MEET always expand fewer states than MM?

A: Not necessarily. MEET and MM may expand a similar number of states (and MEET sometimes expands slightly more due to different tie-breaking or the dynamic heuristic causing different expansion orders). However, MEET's *per-state* cost is dramatically lower because of its efficient TC evaluation. In the brc203d grid map, MEET expands 202 states (111 forward, 91 backward) while MM expands 381 states, but the crucial difference is that MM takes 216 ms vs. MEET's 2.2 ms—a 98× speedup.

Q: What is BAE* and why is it used as a baseline?

A: BAE* (Bidirectional A* with Error functions) is a bidirectional search algorithm from the non-MM family. Unlike MM and MEET, BAE* does not guarantee MMP. It uses a different theoretical framework (must-expand pairs, or MEP) and is considered the state-of-the-art practical Bi-HS algorithm. Siag et al. (2023) recommend BAE* as the default bidirectional algorithm for consistent heuristics. MEET's contribution is showing that an MM-family algorithm can match or beat BAE*'s performance while additionally guaranteeing MMP.

Q: What happens if no solution exists?

A: If both $\textsc{Open}_{\mathcal{F}}$ and $\textsc{Open}_{\mathcal{B}}$ become empty before any TC fires, MEET returns $\emptyset$ (no solution). This happens when the forward and backward searches exhaust all reachable states without finding a connecting path.