An efficient RRT cache method in dynamic environments for path planning

Hyelin Choi

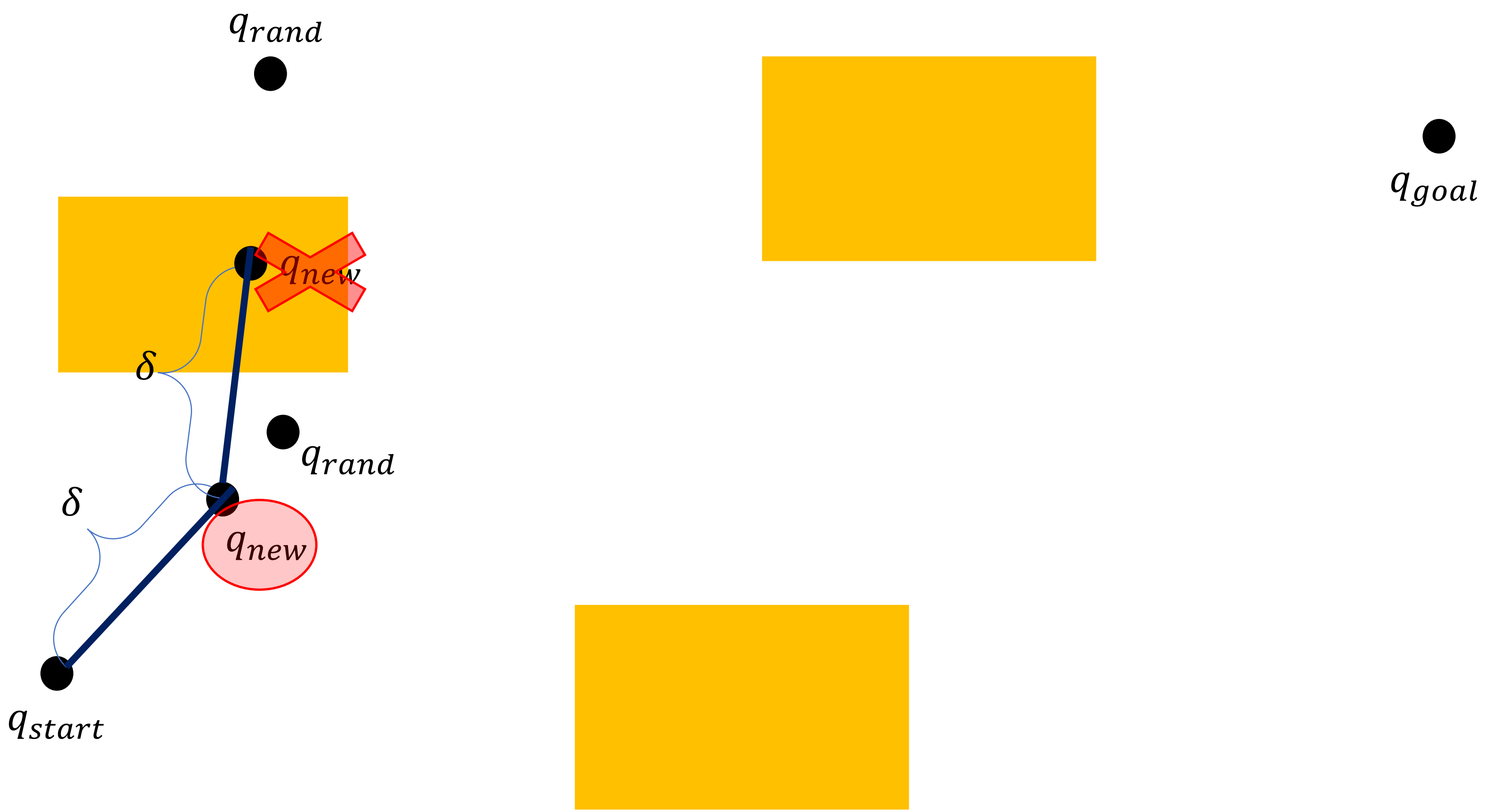
Department of Mathematics
Sungkyunkwan University

January 7, 2024

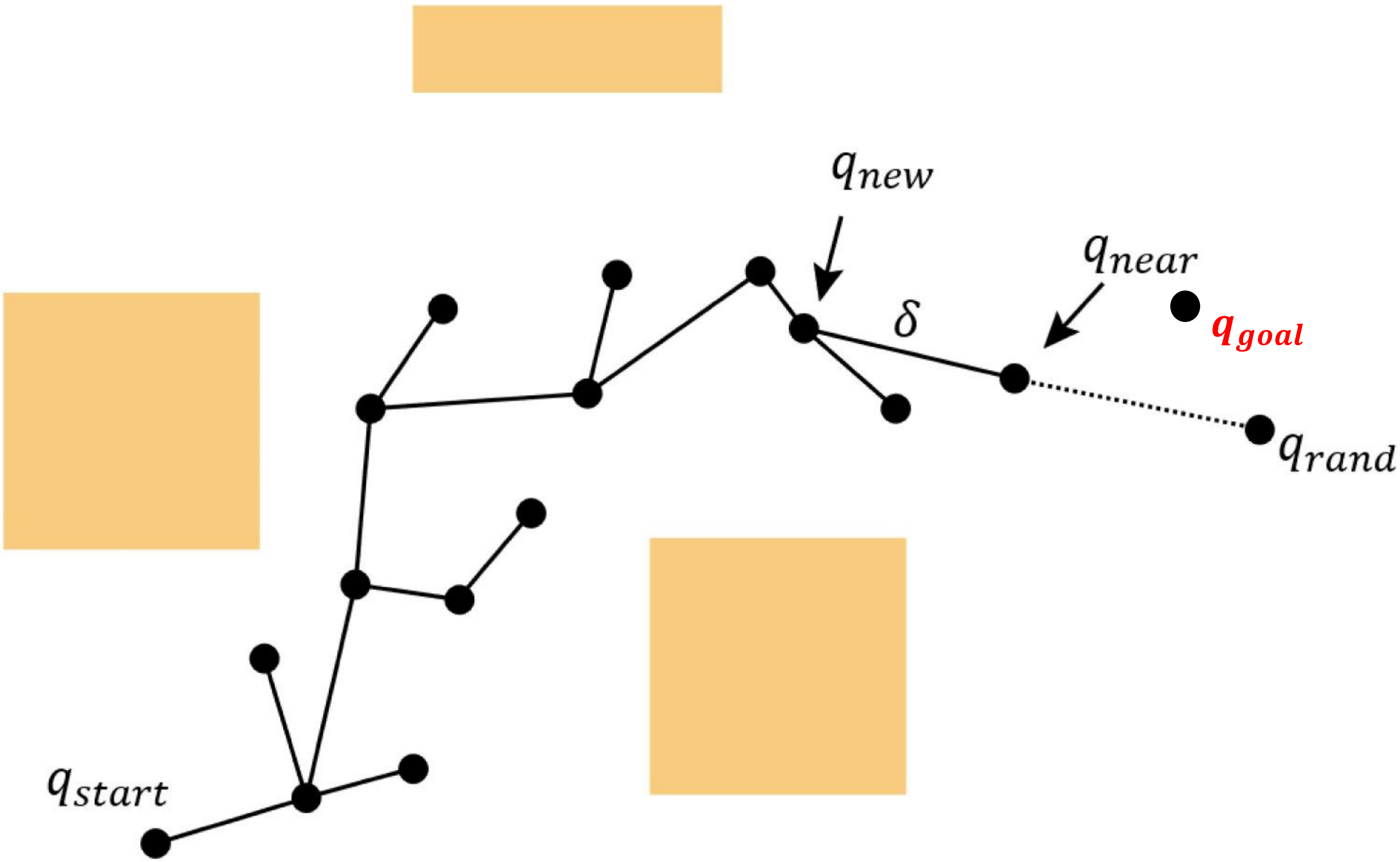
Table of Contents

Table of Contents

RRT



RRT



RRT

Algorithm 1: Basic RRT

Input:

Initial configuration q_{start} q_{goal} S_{obs}
Number of sampling point K
Step size δ

Output:

Search tree T
Vertices of path V

Initialize all Parameters ;
$T = q_{start}$;
for $i = 1$ *to* K **do**
  q_{rand} = Sample();
  $q_{near} \leftarrow$ Nearest($nodes$, q_{rand});
  $q_{new} \leftarrow$ Steer(q_{near}, q_{rand}, δ);
  if *CollisionFree*(q_{near}, q_{new}) **then**
    T.add(q_{new});
    return Advanced;
  if $d(q_{new}, q_{goal}) < Error$ **then**
    return Reached;
  else
    continue ;
final ;
return $T\&V$;

"**Steer**" is calculated according to the ***RRT algorithm***.

"**Advanced**" means that a new node is searched.

"**Reached**" means that the new node reaches the error interval of the goal, and the path planning is completed.

BG-RRT

Algorithm 2: BG-RRT

Input:
Initial configuration q_{start} q_{goal} S_{obs}
Probability of extension P_1 P_2
Number of sampling point K
Step size δ

Output:
Search tree T
Vertices of path V

Initialize all Parameters ;
$T = q_{start}$;
for $i = 1$ *to* K **do**
  q_{rand} = Sample();
  $q_{near} \leftarrow$ Nearest(T, q_{rand});
  p = Rand(1);
  if $p < P_1$ **then**
    q_{rand}
    $q_{new} \leftarrow$ Steer(q_{near}, δ);
    if $P_1 < p < P_2$ **then**
      $q_{new} \leftarrow$ PSteer(q_{near}, q_{goal}, δ);
    else
      $q_{new} \leftarrow$ MixSteer(q_{near}, q_{goal}, q_{rand}, φ, δ);
  if *CollisionFree*(q_{near}, q_{new}) **then**
    T.add(q_{new});
    return Advanced;
  if $d(q_{new}, q_{goal}) < Error$ **then**
    return Reached;
  else
    continue;
final ;
return T&V;

"**Steer**" is calculated according to the ***RRT algorithm***.

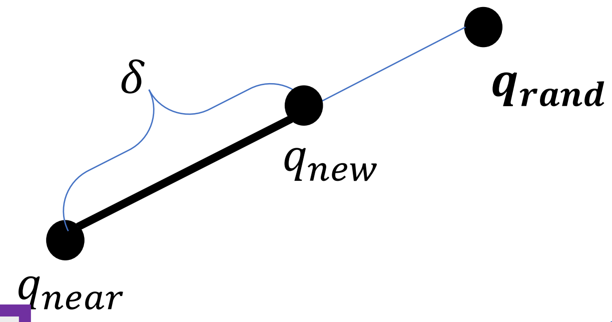
"**PSteer**" is calculated according to the ***potential function-based(P-RRT) algorithm***.

"**MixSteer**" is calculated according to the **bias-goal factor strategy**.

BG-RRT

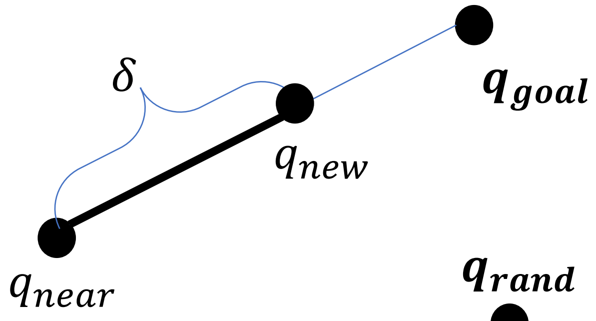
if $p < P_1$ then, **RRT algorithm**

$$q_{new} \leftarrow Steer(q_{near}, q_{rand}, \delta);$$



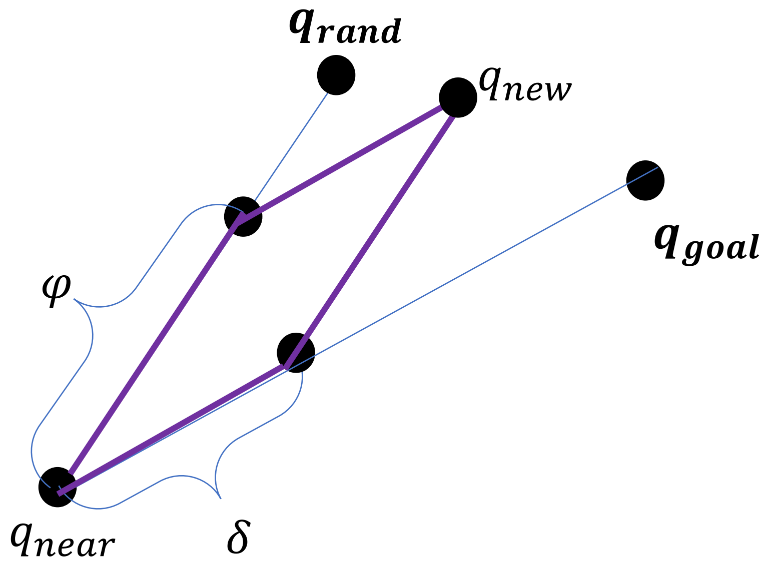
else if $P_1 < p < P_2$ then, **P-RRT algorithm**

$$q_{new} \leftarrow PSteer(q_{near}, q_{goal}, \delta);$$



else **bias-goal factor strategy**

$$q_{new} \leftarrow MixSteer(q_{near}, q_{goal}, q_{rand}, \varphi, \delta);$$



Calculation of q_{new}

In basic RRT algorithm, the new node q_{new} is configured as

$$q_{new} = q_{near} + \delta \frac{q_{rand} - q_{near}}{|q_{rand} - q_{near}|}. \tag{1}$$

In BG-RRT algorithm, the new node q_{new} combined with the bias-goal factor is configured as

$$q_{new} = q_{near} + \delta \frac{q_{rand} - q_{near}}{|q_{rand} - q_{near}|} + \varphi \frac{q_{goal} - q_{near}}{|q_{goal} - q_{near}|}. \tag{2}$$

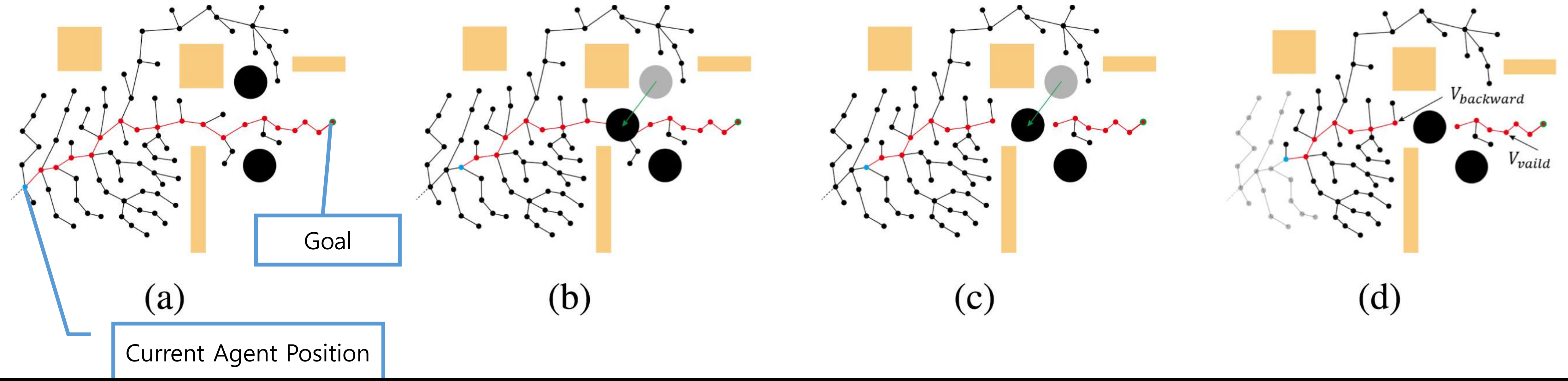
The bias-goal factor is configured as

$$\varphi = \sum_{i=1}^{N} a_i e^{b_i x}, \tag{3}$$

where x is $|q_{rand} - q_{near}|$, a_i and b_i are the indeterminate coefficient of the bias-goal factor and n is proportional to the complexity of a environment.

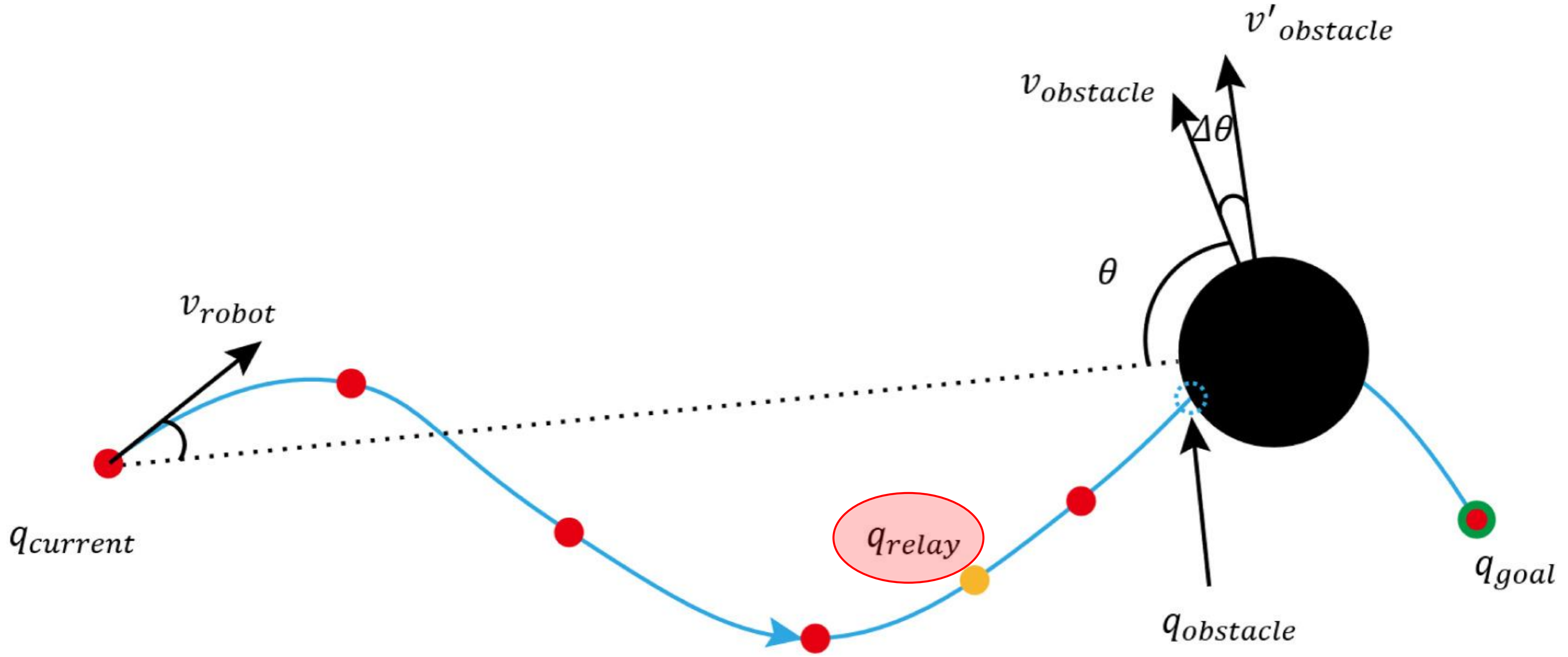
Table of Contents

Pretreatment



An agent moves along with the planned path without changed.	When the previous path is interrupted by a dynamic obstacle, which is the black circle.	Invalid nodes are cut down.	The old nodes are discarded and the path is divided into V_{valid} and $V_{backward}$.

Relay Node



$v'_{obstacle}$
$v_{obstacle}$
$\Delta\theta$
θ
v_{robot}
$q_{current}$
q_{relay}
$q_{obstacle}$
q_{goal}

Relay Node

$q_{relay} = BinarySearch(k) + 1$ where k represents the buffer margin of the relay node which is configuring as

$$k = \frac{d(q_{obstacle}, q_{current}) \cdot v_{max}}{v_{max} - (v_{obstacle} + \Delta v)\cos(\theta + \Delta\theta)}$$

- θ: the angle of the obstacle
- v_{agent}: the speed of the agent
- v_{max}: maximum velocity of the agent
- $q_{current}$: the current position of the agent
- $q_{obstacle}$: the position where the obstacle boundary intersects the path
- $\Delta\theta$: the bias of motion angle
- $v'_{obstacle}$: The velocity of dynamic obstacle.

- The velocity and motion angle are unchanged in each time step. For $i \geq 0$, the anticipation of velocity and motion angle are defined as $v(i+1) = v(i) + \Delta v$ and $\theta\ (i+1) = \theta(i) + \Delta\theta$, where $\Delta v \sim N(\mu_v, \sigma_v^2)$ and $\Delta\theta \sim N(\mu_\theta, \sigma_\theta^2)$.

Relay Node

$q_{relay} = BinarySearch(k) + 1$ where k represents the buffer margin of the relay node which is configuring as

$$k = \frac{d(q_{obstacle}, q_{current}) \cdot v_{max}}{v_{max} - (v_{obstacle} + \Delta v)\cos(\theta + \Delta\theta)}$$

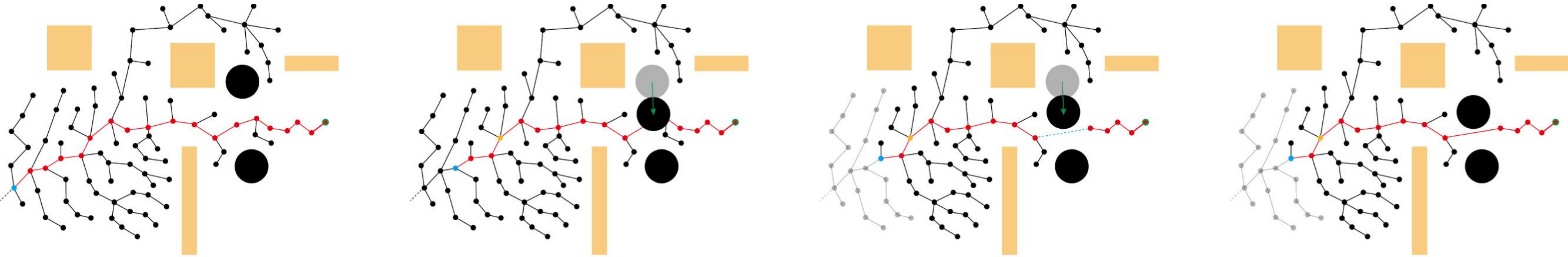
- θ: the angle of the obstacle
- v_{agent}: the speed of the agent
- v_{max}: maximum velocity of the agent
- $q_{current}$: the current position of the agent
- $q_{obstacle}$: the position where the obstacle boundary intersects the path
- $\Delta\theta$: the bias of motion angle
- $v'_{obstacle}$: The velocity of dynamic obstacle.

- The velocity and motion angle are unchanged in each time step. For $i \geq 0$, the anticipation of velocity and motion angle are defined as $v(i+1) = v(i) + \Delta v$ and $\theta\,(i+1) = \theta(i) + \Delta\theta$, where $\Delta v \sim N(\mu_v, \sigma_v^2)$ and $\Delta\theta \sim N(\mu_\theta, \sigma_\theta^2)$.

BinarySearch(k)

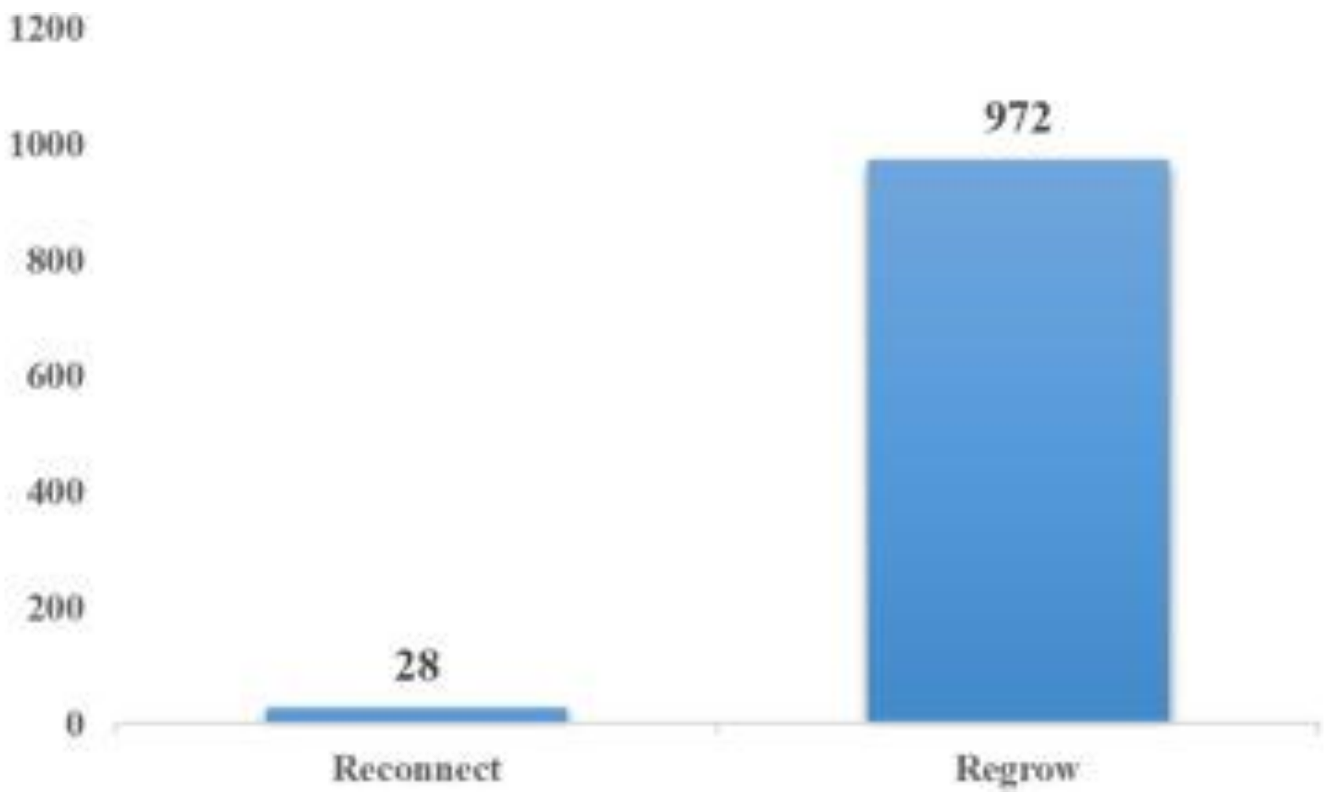
Table of Contents

Reconnect



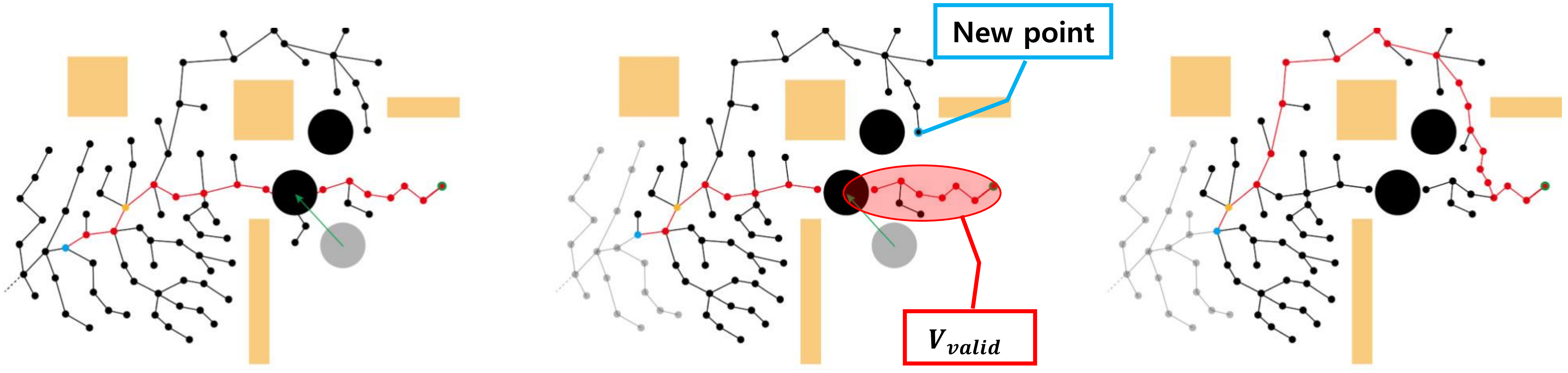
(a) (b) (c) (d)

The agent is moving along with the path in an initial environment.	When an interrupted path is detected, the relay node is generated.	The invalid node is deleted.	A new path is connected to repair in a single step.



The number of Regrow strategy is significantly more than the Reconnect strategy.

Regrow



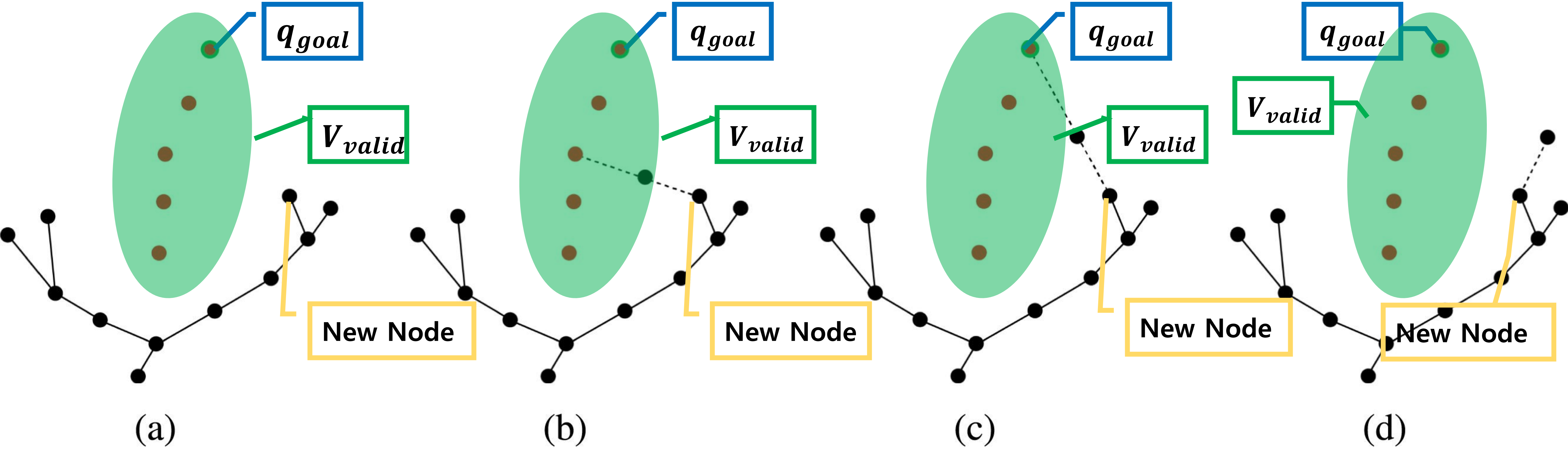
(b)	(c)	(d)
The relay node is used.	The black point with blue is set as a new point and V_{valid} is set a new goal.	If only the error is satisfied, a new tree will extend to any point in V_{valid}.

Table of Contents

Waypoint Cache

- Waypoint Cache method is introduced to improve the path repairing efficiency.
- Waypoint Cache method makes use of **potential cache information**.

Waypoint Cache



	If $0 < p < P_{way}$, a new node will extend along to the direction of $V_{valid}(i)$.	If $P_{way} < p < P_{way} + P_{goal}$, a new node will extend along to the direction of q_{goal}.	If $p > P_{way} + P_{goal}$, the node will extend in a random direction.

Waypoint Cache

Algorithm 3: Waypoint Cache

Input:

The probability towards the waypoint P_{way}
The probability towards the random P_{rand}
The probability towards the goal P_{goal}

Output:

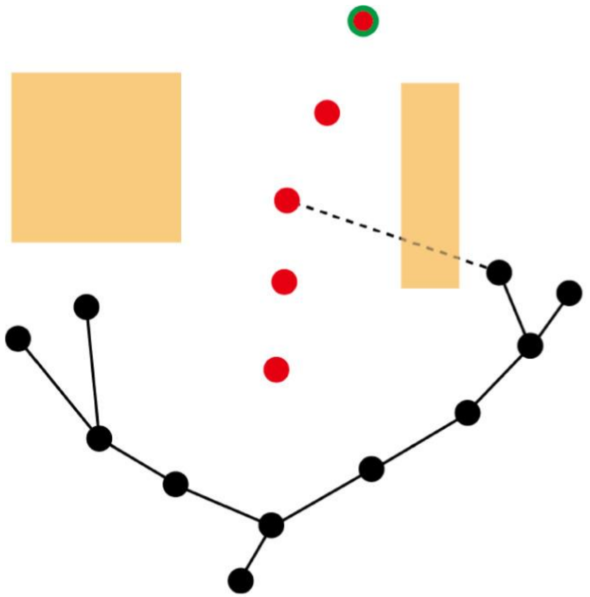
RandomState()
Cache number i

Initialize all Parameters ;
while *ChooseGoal()* **do**
  p = RandomFloat in [0, 1.0] ;
  i = RandomInt in [1,n] ;
  if $0 < p < P_{way}$ **then**
    return P_{way};
    if $P_{way} < p < P_{way} + P_{goal}$ **then**
      return P_{goal};
  else
    return P_{rand};
return RandomState()&i;

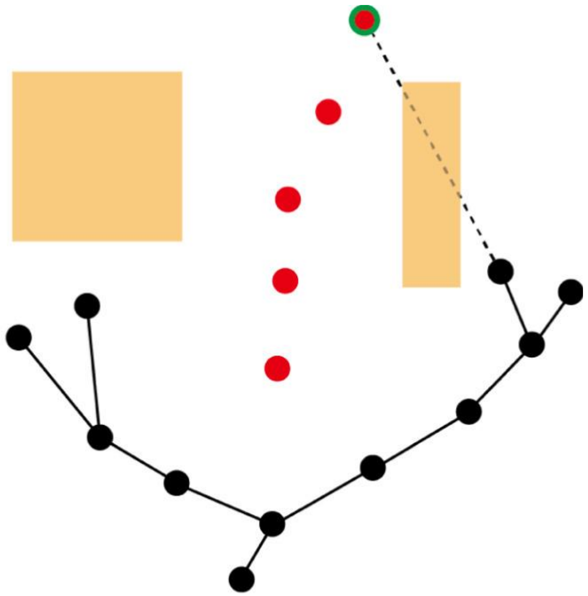
- P_{goal} : the growth probability towards the goal.
- P_{way} : the growth probability towards the V_{valid}.
- p ($p \in (0, 1)$) is random number.
- i ($i = 1, \cdots, n$)is random number.
- n : the number of node in V_{valid}.

Waypoint Cache Problem

- It is mainly caused that the path from the new q_{start} to V_{valid} is blocked.
- P_{way}, P_{goal} have significant diversity between different algorithms and unstructured environments.
- The point selected from V_{valid} randomly is not optimal for dynamic obstacle avoidance.

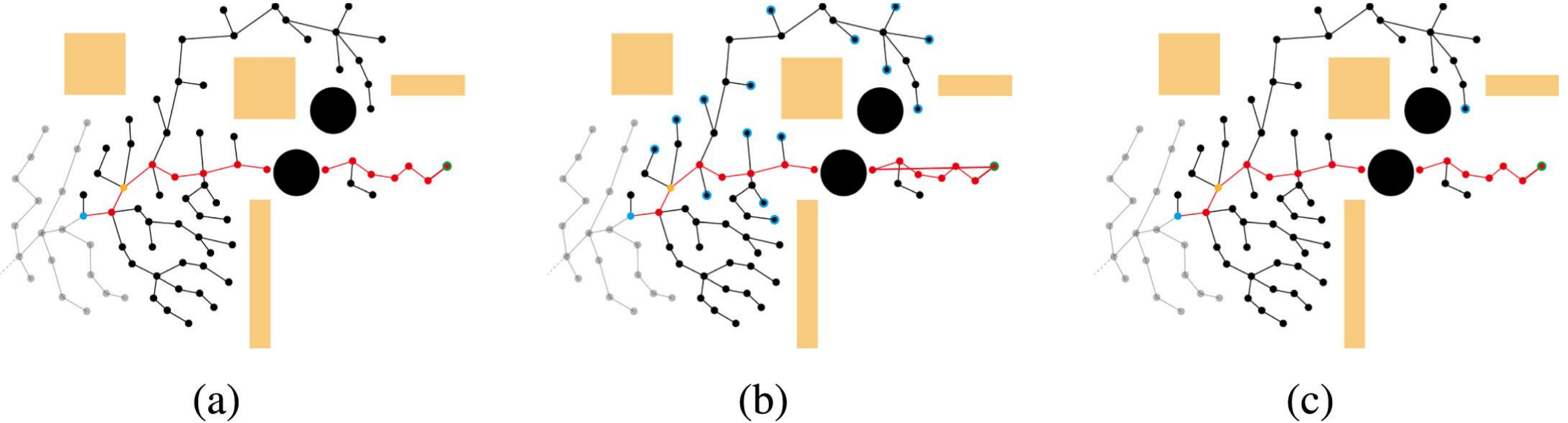


(a)



(b)

Efficient and Optimal Waypoint Cache(EOWC)



(a)	(b)	(c)
Detect dynamic obstacle avoidance and wait for the production of the relay node.	The end nodes of T are selected as $N = [N_1, N_2, \cdots, N_m]^T$ with a black and blue point after q_{relay}.	The minimum cost node is selected.

Efficient and Optimal Waypoint Cache(EOWC)

Algorithm 4: EOWC

Input:
Vertices of path V
Search tree T

Output:
New goal $V_{\cancel{n}}$ a
New start $N_{\cancel{m}}$ b

Initialize all Parameters ;
while *Regrow* **do**
  $V_{vaild} \& V_{backward} \leftarrow$ Pretreatment(T) ;
  $N_{\cancel{m}}$ a $\leftarrow \boxed{\text{Select}(V_{vaild}, V_{backward})}$;
  $V_{\cancel{n}}$ b $\leftarrow \boxed{\text{CostLeast}(V_{vaild}, N_{\cancel{m}})}$ a ;
  return $N_{\cancel{m}}$ a & $V_{\cancel{n}}$ b ;

Find a **start node** $\boldsymbol{N_a}$ among $N_1, N_2, \cdots N_m$.

Find a **goal node** $\boldsymbol{V_b}$ among $V_1, V_2, \cdots V_m$.

$Select()$

$$N_{\underset{a}{\cancel{m}}} \leftarrow Select(V_{valid}, V_{backward}) \tag{1}$$

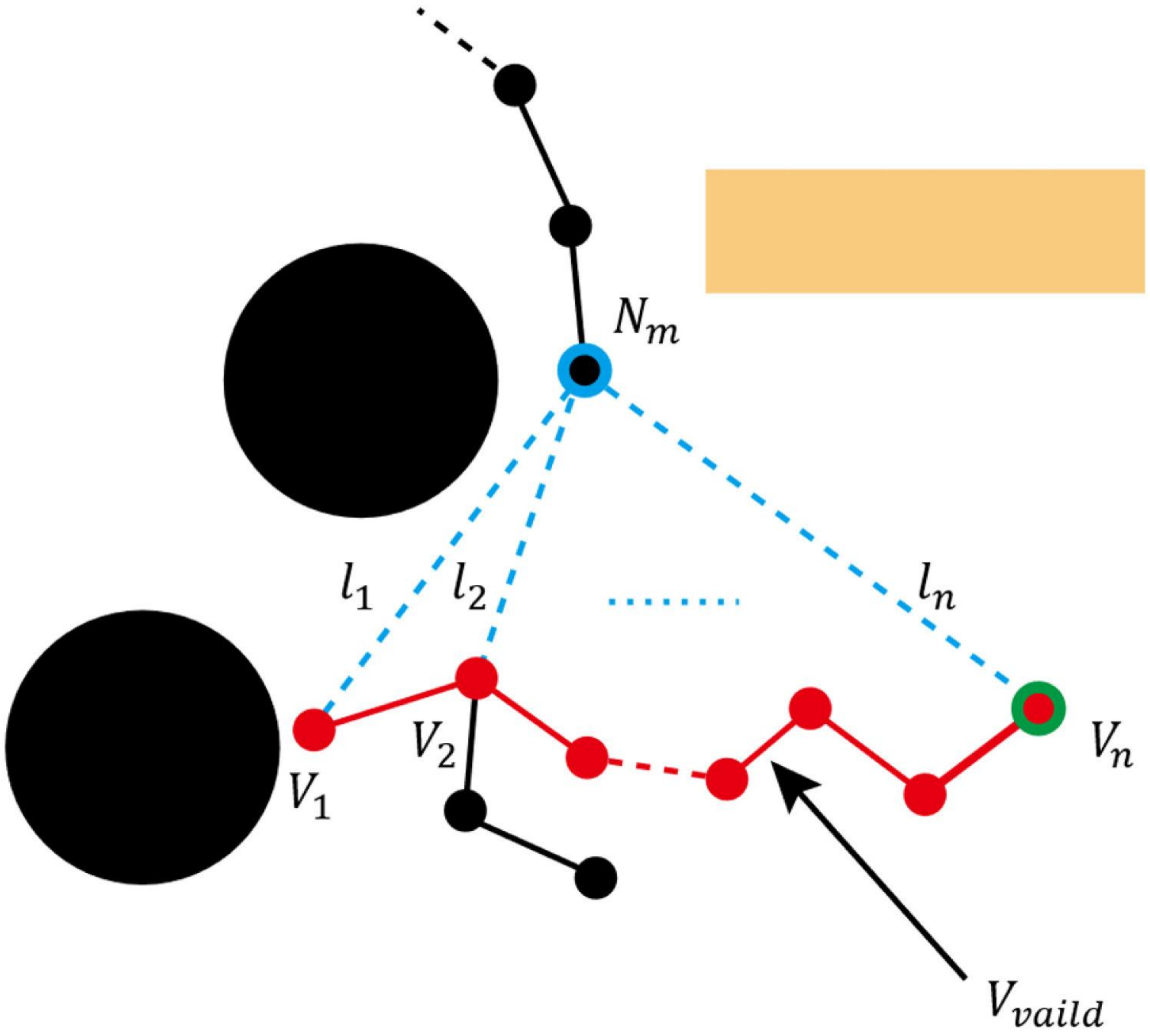
In order to search the least cost of $N = [N_1, N_2, \cdots N_m]^T$ to $V_{valid} = [V_1, V_2, \cdots, V_n]^T$, the $Select()$ function is defined as

$$S(m) = \begin{cases} \boxed{d(m),} & \text{if } l_i \cap S_{obs} = \emptyset \\ d(m) + \xi, & \text{otherwise} \end{cases},$$

$$d(m) = \begin{cases} [d(N, V_1)]_{m\times 1}, & n = 1 \\ \begin{bmatrix} d(N, V_1) \\ d(N, V_2) \end{bmatrix}_{2m\times 1}, & n = 2 \\ \boxed{\begin{bmatrix} d(N, V_1) \\ d(N, V_n) \\ d(N, l_{V_1 V_n}) \end{bmatrix}_{3m\times 1}, \quad n > 2} \end{cases} \tag{2}$$

where $l_{V_1 V_n}$ is lined by V_1 and V_n and ξ is defined as obstacle safety factor according to the collision detection.

$Select()$



Efficient and Optimal Waypoint Cache

Algorithm 4: EOWC

Input:

Vertices of path V

Search tree T

Output:

New goal $V_{\cancel{n}}$ a

New start $N_{\cancel{m}}$ b

Initialize all Parameters ;

while *Regrow* **do**

$V_{vaild} \& V_{backward} \leftarrow$ Pretreatment(T) ;

$N_{\cancel{m}}$ a $\leftarrow$ $\boxed{\text{Select}(V_{vaild}, V_{backward})}$;

$V_{\cancel{n}}$ b $\leftarrow$ $\boxed{\text{CostLeast}(V_{vaild}, N_{\cancel{m}})}$;

a

return N_m & V_n ;

Find a **start node N_a** among $N_1, N_2, \cdots N_m$.

Find a **goal node V_b** among $V_1, V_2, \cdots V_m$.

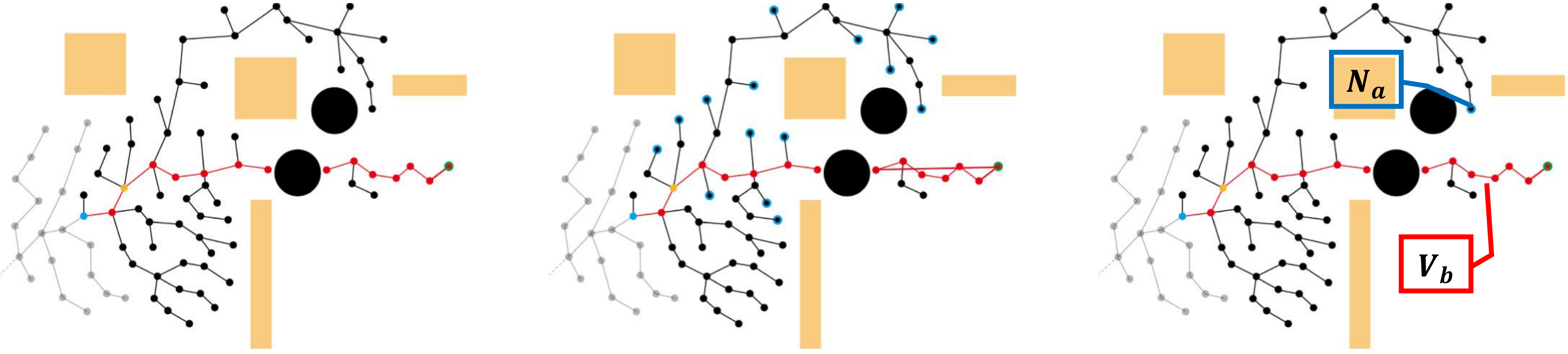
$CostLeast()$

The $CostLeast()$ is defined as

$$C(V_i) = \frac{\rho(N_m, V_i)}{\sum_{i=1}^{N} \rho(N_m, V_i)} e^{\lambda \cdot d(N_m, V_i)} \tag{3}$$

where $\rho = [\rho_1, \rho_2, \cdots, \rho_n]^T$ is the curvature of discrete point, and $\rho(N_m, V_i)$ is the curvature of $N_m, V_i\,(i = 1, 2, \cdots, n-1)$, and V_n. Specially, $\rho(N_m, V_n)$ is defined as 1.

Efficient and Optimal Waypoint Cache



(a) (b) (c)

Detect dynamic obstacle avoidance and wait for the production of the relay node.	The end nodes of T are selected as $N = [N_1, N_2, \cdots, N_m]^T$ with a black and blue point after q_{relay}.	The minimum cost node is selected.

EBG-RRT

Algorithm 5: EBG-RRT

Input:
  Update configuration S
  Execution trajectory τ
  Search tree T

T is determined by ***BG-RRT***.

Initialize all Parameters ;
Init movement ;
while S_{obs} *changes* **do**
  if $\tau \bigcap S_{obs} = \emptyset$ **then**
    return ContinueMovement;
  else
    $q_{init} \leftarrow q_{current}$;
    $q_{relay} \leftarrow \text{RelayNode}(S_{obs}, q_{current})$;
    $V_{vaild} \& V_{backward} \leftarrow \text{Pretreatment}(T)$;
    if $SingleStep \leftarrow ture$ **then**
      $\tau \leftarrow \text{Reconnect}(V_{vaild}, V_{backward})$;
      return τ ;
    else
      $N_m \& V_n \leftarrow \text{EOWC}(T)$;
      $\tau \leftarrow \text{Regrow}(N_m, V_n)$;
      return τ ;

Thank you for listening.