

# 外文翻译

毕业设计题目： 基于机器学习的车位规划

原文 1: Spatial planning of urban communities via deep  
reinforcement learning

译文 1: 基于深度强化学习的城市社区空间规划

原文 2: Puzzle-based parking

译文 2: 基于谜题的停车

# Spatial planning of urban communities via deep reinforcement learning

Yu Zheng, Yuming Lin, Liang Zhao, Tinghai Wu, Depeng Jin, Yong Li

Tsinghua University, Beijing, P. R. China.

liyong07@tsinghua.edu.cn

**Abstract:** Effective spatial planning of urban communities plays a critical role in the sustainable development of cities. Despite the convenience brought by geographic information systems and computer-aided design, determining the layout of land use and roads still heavily relies on human experts. Here we propose an artificial intelligence urban-planning model to generate spatial plans for urban communities. To overcome the difficulty of diverse and irregular urban geography, we construct a graph to describe the topology of cities in arbitrary forms and formulate urban planning as a sequential decision-making problem on the graph. To tackle the challenge of the vast solution space, we develop a reinforcement learning model based on graph neural networks. Experiments on both synthetic and real-world communities demonstrate that our computational model outperforms plans designed by human experts in objective metrics and that it can generate spatial plans responding to different circumstances and needs. We also propose a human-artificial intelligence collaborative workflow of urban planning, in which human designers can substantially benefit from our model to be more productive, generating more efficient spatial plans with much less time. Our method demonstrates the great potential of computational urban planning and paves the way for more explorations in leveraging computational methodologies to solve challenging real-world problems in urban science.

## 1 Methods

### 1.1 Problem formulation

We formulate the problem of community spatial planning as a sequential Markov decision process (MDP), an interactive process between the planning agent and the environment, in which the agent observes the ‘state’ (the current conditions of the community), takes an ‘action’ (placement of an urban functionality) at each step and receives ‘reward’ (effect of the planned result) signaled by the environment, which undergoes a ‘transition’ (changes of the layout) according to the agent’s action. We utilize DRL to learn an effective policy that maps states to actions with a parameterized neural network. The neural network is optimized towards higher spatial efficiency through massive training under the MDP, with millions of training samples of the 4-tuple (state, action, reward and transition). As illustrated in Extended Data Fig. 1, our MDP is composed of two consecutive stages:

- Land-use planning. Given the initial road conditions, the agent places functionality blocks one at a time, either near existing roads or near boundaries of previously placed land use. After all the functionalities and open spaces are allocated, a reward regarding the efficiency of land use is returned to the agent, which treats different land use as an integrated system. The final land-use plan becomes the initial condition of road planning.
- Road planning. Boundaries of planned land use are viable locations for road construction. The agent builds roads iteratively, turning one boundary into a road segment at a single step. Stopped at a predefined termination step, a reward considering the transportation efficiency is returned to the agent.

The reward is only calculated at the last step of each stage to summarize the performance of land-use planning and road planning, respectively, and all intermediate steps receive a reward of 0. We define the reward for land use and road layout based on the 15-minute-city concept, which emphasizes the spatial efficiency to facilitate active transportation such as walking and cycling instead of automobiles. The two reward terms are calculated as follows:

$$r_L = \alpha Service + Ecology, \quad (1)$$

$$r_R = Traffic, \quad (2)$$

where Service measures the community life-circle index in the 15-minute city, Ecology measures the coverage of green space and parks, Traffic is a combination of road density and connectivity (Methods) and  $\alpha$  serves as a hyper-parameter indicating the weight of service performance in the land-use reward. With the calculated reward values, we use proximal policy optimization to update the parameters of value and policy networks. We first train the agent for the land-use planning task until convergence and then train the agent to build roads with the optimal land-use plan obtained in the first stage. After two stages of training, the AI agent is able to design communities with an efficient spatial layout of both land use and roads.

**Graph model.** Different from previous Go and chip design tasks, urban planning is more challenging because of its much larger degrees of freedom in the problem form. Specifically, the conditions of previous tasks are regular, for example, placing stones on a  $19 \times 19$  board or placing rectangular macros onto a grid chip canvas, which can be represented by pixels (raster). By contrast, the conditions for community spatial planning are diverse and irregular because the road corners and land blocks are usually not orthogonal. To accurately describe urban geographic elements, including land blocks (L), segments of roads and land-use boundaries (S) and junctions between roads and land-use boundaries (J), we use vector representations, which have been proven to have substantial advantages over raster representations in urban planning, and consist of the following three geometries:

- ‘Polygon’ that describes a vacant land to be planned (for example, L1 in the right part of Extended Data Fig. 2a) or an already planned land block (for example, L2 in the right part of Extended Data Fig. 2a) with the coordinates of the land boundary;
- ‘LineString’ that represents a road segment (for example, S3 in the right part of Extended Data Fig. 2a) or a boundary edge of a land block (for example, S9 in the right part of Extended Data Fig. 2a) with the coordinates of the start and end points; and
- ‘Point’ that stands for the junctions between roads and landblock boundaries (for example, J2 and J7 in the right part of Extended Data Fig. 2a) with their coordinates.

We transform all geographic elements into the above three categories of geometries and then represent the whole community as a graph, in which nodes are the geometries and edges stand for the spatial contiguity relationship between these geometries, that is, two nodes are connected if the underlying two geometries touch each other. Each node stores its geographic information as the node features, including the type, coordinates, width, height, length and area of the geometry. In this way, spatial planning can be transformed as a problem of making choices on a dynamic graph (Extended Data Fig. 2), in which the graph evolves according to the agent’s actions.

In the land-use planning task, the agent selects one L–J edge that connects a vacant land and a junction, placing a given functionality at the location specified by the corresponding L and J (Extended Data Fig. 2a).

In each step, the topology of the contiguity graph changes because the newly placed functionality generates new nodes and edges. New nodes include the new functionality itself, its boundaries, new junctions and split segments. New edges indicate the newly established spatial contiguity. Similarly, in the road planning task, the agent selects one S node that is currently a boundary and constructs it as a road segment (Extended Data Fig. 2b). Although the topology remains the same, the graph’s attributes alter because the selected node’s type changes from boundary to road. Through the problem reformulation with the graph model, we can now handle the irregular urban blocks and unify the two seemingly distinct stages of land use and road planning on one single graph.

**Action space design.** Another major challenge of urban planning is the huge action space, which is almost infinite in the original continuous space, and still too large in the reduced discrete graph space. The contiguity graph continues to grow as we place a functionality at each step, resulting in a large graph with thousands of nodes and edges. A typical spatial plan of a 2 km by 2 km community can take a total number of 100 planning steps in each stage, and the contiguity graph can have 4,000 edges and 1,000 nodes, which makes the action space  $4,000^{100}$  and  $1,000^{100}$  for the two stages, respectively. In addition, valid actions are extremely sparse in the space, and a substantial portion of actions is of low quality and will lead to unreasonable results, such as placing a facility in the center of a vacant land without connecting roads. Therefore, it is crucial to reduce the action space and avoid unreasonable actions.

To address this challenge, we propose a general DRL framework in which an intelligent agent perceives and makes decisions in a reduced graph space, and the environment handles urban elements in the original geographic space and generates graph states according to the geographic spatial layout. Meanwhile, we decompose the entire action space into a Cartesian product of three sub-spaces, including what to plan, where to plan and how to plan, and let the DRL agent focus on the core issue of where to plan. The first sub-space of what to plan can be eliminated by fixing the planning order of different land-use types through domain knowledge, allowing land-use types that are more dependent on the initial road network to be planned earlier (Methods). To avoid apparently improper actions in where to plan, we impose planning constraints on the agent’s actions, with an action mask that blocks out unreasonable options, that is, only L–J edges and S nodes are candidates for the two planning stages, respectively. After selecting one L–J edge for a given land-use type, the functionality is placed in the corresponding land block (L node) at the location of the corresponding junction (J node), whose shape and size are determined by predefined rules that maximize the reuse of existing roads and boundaries (Methods); thus, the last sub-space of how to plan is effectively eliminated. Through these designs, we narrow the action space to a solvable scale and filter out most unreasonable actions, enabling efficient optimization for DRL algorithms. In summary, the original problem of spatial planning is successfully transformed into a standard sequential decision-making process on a graph with moderate action space.

**Framework.** After the above problem reformulation and action space design, we propose a DRL framework in which an AI agent learns to lay out land use and roads by interacting with the spatial planning environment, as illustrated in Extended Data Fig. 3. The sequential MDP (Extended Data Fig. 3e,f) contains the following key components:

- States summarize the current spatial plan with the previously introduced contiguity graph containing rich node features, and other information, such as statistics of different land use types.
- Actions indicate the locations to place the current land use or construct a new road segment, which are transformed from the selected edges or nodes in the contiguity graph.
- Rewards are 0 for all intermediate steps, except for the last step in each stage, in which it evaluates the spatial efficiency of land use and roads.

- Transitions describe the changes of the layout given the selected location, and the transitions occur in both the original geographic space (new land use and road on the map) and the transformed graph space (new topology and attributes of the graph).

At each step, the agent represents the state by encoding the graph with a GNN. Via multiple message passing and non-linear activation layers, the GNN state encoder generates effective representations of edges, nodes and the whole graph (Extended Data Fig. 3a), which will be leveraged by the value and policy networks (Extended Data Fig. 3b–d). Specifically, because choosing locations for land use is equivalent to selecting edges on the graph, the land-use policy network takes the edge embeddings and scores each edge with an edge-ranking MLP, as shown in Extended Data Fig. 3b. The obtained score for each edge indicates the sampling probability of the corresponding edge, which is returned to the environment and becomes the probability of placing the land use at the location specified by that edge. Similarly, in road planning, the road policy network takes node embeddings and scores each node with a node-ranking MLP (Extended Data Fig. 3d), outputting the probability of choosing one land block boundary and building it as a road segment. Finally, the value network takes in the graph embedding that summarizes the whole community and predicts the planning rewards with a fully connected layer (Extended Data Fig. 3c). To master the skills of spatial planning, millions of spatial plans are accomplished by the proposed model to search the large solution space during the training process, which is utilized as real-time training data to update the parameters of the neural network.

## 1.2 Detailed methodology

**Framework.** As introduced in the paper, we use vector geometries including Polygon, LineString and Point to describe urban geographic elements. Specifically, there are ten types of land blocks that are represented as Polygon, including the initial vacant land to be planned, and nine different functionality types, which are residential (RZ), school (SC), hospital (HO), clinic (CL), business (BU), office (OF), recreation (RE), park (PA) and open space (OP). In addition, there are two types of segments (roads and land-use boundaries) that are represented by LineString and one type of junction (intersections between roads and land-use boundaries) that is represented by Point. Therefore, a community is faithfully represented by a table of geometries, in which each row is a geographic element with three columns of ID, type and geometry. Initial conditions of a community consist of all the original land blocks, roads and intersections, whose accurate coordinates are recorded by their corresponding geometries in the table of geometries. In the synthetic grid community, we experiment on a basic community with a size of 2.4 km by 2.4 km, containing 16 rectangular vacant lands, 40 horizontal or vertical initial road segments and 25 road intersections, as shown in the first step of Extended Data Fig. 1. In the real-world community, we replicate the road network of HLG and DHM communities in Beijing from OpenStreetMap using OSMnx30 and geopandas, reserve residential blocks and leave other areas as vacant land to be renovated. Finally, we obtain two communities of around 4 km<sup>2</sup> as shown in Fig. 1a and Supplementary Fig. 11a.

**Planning needs and requirements.** Before carrying out the actual spatial planning, we need to determine the planning needs and requirements, which serve as the configuration of the planning environment. The planning need describes the amount that each land-use type has to achieve, either in area or in number, for example, residential blocks of 50% community area and three hospitals. Meanwhile, we also have requirements on the minimum area (in square meters) of each planned block; for example, the area of one school is at least 10,000 m<sup>2</sup>. Supplementary Table 3 shows an example of the planning needs and requirements for a community, in which 15% of the community area needs to be planned as parks; thus, it serves as a green community. Only spatial plans that satisfy all the needs and requirements are considered as successful

episodes and reserved as training samples, and those failed episodes are discarded. In our framework, the planning needs and requirements are configurations for the environment, making our model highly flexible in generating spatial plans. Specifically, once we obtain a well-trained model under one configuration, we can simply change the configuration and directly perform model inference without re-training to generate plans for different planning needs and requirements, such as the community plans of different service supplies in Fig. 1c,d.

**Planning order of land-use types.**As introduced in the paper, in order to reduce the huge action space, we fix the planning order of different land-use types based on domain knowledge and make the agent focus on the core task of selecting locations. Because feasible locations are next to existing junctions, land use that is planned earlier will be closer to initial roads with more convenient traffic. Therefore, we first plan those facilities that depend more on roads, including hospitals (clinics), schools and recreation. Meanwhile, at later steps of land-use planning, the shape of feasible vacant lands tends to be more irregular and fragmented, which is not suitable for residential blocks that usually occupy a whole plot of land; thus, we distribute residential blocks after planning the above road-dependent facilities. Finally, we arrange those land-use types that are not much demanding in land shapes. After all the planning needs are satisfied, the remaining vacant lands are assigned as open spaces. In summary, the planning order in our framework is fixed as follows: hospital, school, clinic, recreation, residential, park, office, business and open space. Letting the agent determine the order of land-use types may be an alternative approach. However, it will make the problem much more complicated, as the action space is increased drastically. In practice, our fixed order generates sound spatial plans.

**Land cutting.**In land-use planning, the environment receives the action from the agent, which is the selected L–J edge, and cuts a new land from the corresponding land block (L node) at the location of the corresponding junction (J node). We develop a rule-based system with expert knowledge incorporated to determine the shape and size of the new land. The rule-based system is roughly composed of three steps: (1) Determine the relationship between J and L, such as in the middle of a road or at the corner. (2) Determine the reference line along existing boundaries from junction J, which can be I-shape, L-shape and U-shape. (3) Determine the length of inward extension from the reference line into the block L, forming the final sliced new land. The three steps are conducted according to expert knowledge, in order to meet the planning requirements and fit the current plan as closely as possible.

**State.**Our state contains three parts: (1) urban contiguity graph, (2) current object to be placed and (3) community statistics. We construct a graph to represent the current community information as illustrated in Extended Data Fig. 2, in which nodes are urban geographic elements and edges indicate the spatial contiguity relationship. We compute rich geographic attributes as node features, including the type, coordinates, area, length, width and height of the underlying urban element. The edges are represented by a sparse adjacency matrix. As for the current object to be placed, its type is determined by the environment according to planning needs and planning order, that is, the environment will traverse the planning order and transit to the next type if the planning needs for the previous type have been satisfied. We treat the current object as a virtual isolated node, with its type feature provided by the environment and other node features left as default values. Lastly, community statistics include the area and count of different land-use types in the current plan, as well as the planning needs, which summarize the current conditions and the progress of spatial planning.

**Action.**As illustrated in Extended Data Fig. 2a, land-use planning is reformulated as a sequential MDP in which the agent selects an edge in a dynamic graph. Therefore, the action space for land-use planning is the probability distribution of choosing from  $N$  edges, and we sample from this distribution to obtain the action. Similarly, road planning is a sequential MDP of choosing nodes as shown in Extended Data Fig. 2b; thus, the action space for road planning is the probability distribution over  $M$  nodes, which is sampled

to generate the node selection action. In addition, as introduced previously, we impose constraints on the action space; for example, the agent can only select L–J edges (between vacant lands and junctions) and S nodes (land-use boundaries) to avoid unreasonable spatial plans. Thus, we calculate a mask in each step that indicates feasible options, and the probability distribution will be multiplied by the mask, allowing only feasible edges or nodes to be sampled as actions.

**Policy and value networks.** As shown in Extended Data Fig. 3b–d, we develop separate policy networks to take actions in the policy and value stages, respectively, as well as a value network to predict the performance of spatial plans. The three networks share the same state encoder to obtain state representations, taking full advantage of the GNN. Policy networks generate the probability distribution by scoring the edges and nodes in the graph, and then sample from this distribution to take actions. Meanwhile, the value network evaluates the whole graph to predict spatial efficiency, providing feedback for the community plan. In this section, we introduce the detailed design of the three networks.

**Land-use policy network.** In land-use planning, the agent places the current object at the location specified by the selected edge. The effect of edge selection is related to both the edge and the current object; for example, placing a hospital next to an already planned hospital may lead to low service efficiency. Therefore, the land-use policy network considers both the edge and the current object as input. As shown in Extended Data Fig. 3b, a feed-forward network, which is an edgeranking MLP, is developed to score each edge:

$$s(e_{ij}) = FF_{land}(e_{ij}^L || v_c || e_{ij}^L - v_c || e_{ij}^L \cdot v_c), \quad (3)$$

here the difference and the inner product of  $e_{ij}^L$  and  $v_c$  are also concatenated to emphasize the relationship between the current object to be planned and those already planned land use. The scores are converted to a probability distribution over all edges using softmax:

$$Prob(e_{ij}) = \frac{e^{S(e_{ij})}}{\sum_{s,t \in E} e^{S(e_{st})}}, \quad (4)$$

which is sampled to select an edge.

**Road policy network.** In road planning, the agent selects one boundary node and plans a road at its location. Different from that of land-use planning, the topology of the graph is stable with no new nodes to be added. Thus, there is no need to include the current object. Meanwhile, the road policy network takes node embeddings as input, which already contain neighbor geographic information through message passing of GNN. As in Extended Data Fig. 3d, another node-ranking feed-forward MLP is adopted to score each node:

$$s(v_i) = FF_{road}(v_i). \quad (5)$$

The score is also transformed to probability with a softmax operator:

$$Prob(v_{ij}) = \frac{e^{S(v_{ij})}}{\sum_{j \in N} e^{S(v_j)}}, \quad (6)$$

and we sample from this probability distribution to select one node.

**Value network.** As shown in Extended Data Fig. 3c, we develop a value network to judge the current planning situations and predict the planning performance. Because it is an overall evaluation of the entire community, we take the graph-level representation as the input of the value network. Meanwhile, we also include community statistics. Specifically, we concatenate graph representations and the statistics embedding,

and adopt a fully connected layer to predict the performance:

$$\hat{v} = fa(g^L || h_s), \quad (7)$$

where  $\hat{v}$  is the estimated value of the current plan.

**Reward.** We train the policy networks to optimize the efficiency of spatial layout, with respect to service, ecology and traffic. As in equations (1) and (2) of this paper, we define reward functions that give a comprehensive evaluation of the above metrics. Meanwhile, the reward values can be quickly computed within tens of milliseconds given a spatial plan, making it possible to collect large-scale samples for training DRL models. In this section, we introduce how the three metrics are calculated. It is worth noting that our framework is flexible and can be extended to include more metrics in the reward.

**Service.** We adopt the concept of 15-minute life circle, which requires that basic services of the community be reachable for residents within 15 min by walking or cycling. Specifically, as demonstrated in Fig. 1b,c, we consider five different basic services, each of which is related to one or two facilities, that is, education (school), medical care (hospital, clinic), working (office), shopping (business) and entertainment (recreation). Therefore, the 15-minute life circle means that the distances between facilities and residential zones need to be less than the walking distance of 15 min, which is set as 500 m in our experiments. We define the service metric as the proportion of accessible services within 500 m, and the metric is averaged for all residential zones. Formally, given a community spatial plan  $p$ , the service metric is calculated as follows:

$$d(i, j) = \min\{EucDis(RZ_i, FA_1^j), \dots, EucDis(RZ_i, FA_{n_j}^j)\}, \quad (8)$$

$$Service_i = \frac{1}{5} \sum_{j=1}^5 \mathbb{I}[d(i, j) < 500], \quad (9)$$

$$Service = \frac{1}{n_{RZ}} \sum_{i=1}^{n_{RZ}} Service_i, \quad (10)$$

where  $EucDis$  is the Euclidean distance,  $d(i, j)$  is the minimum distance for the  $i$ th residential zone  $RZ_i$  to access the  $j$ th service that is provided by facility  $FA^j$  and  $n_j$  is the total number of facilities  $FA^j$ .  $Service_i$  is the 15-min life-circle metric for the  $i$ th residential zone, and we average over all  $n_{RZ}$  residential zones to obtain the final service metric for the whole community. This service metric guides the agent to arrange facilities in a more decentralized way and close to residential zones, which is critical for increasing the ability of community services.

**Ecology.** The ecology of a community is important to the physical and mental health of residents; thus, we include an ecology metric that measures the layout efficiency of parks and open spaces. In general, parks and open spaces serve the residents who live in the neighborhood, and we hope they can serve as many residential areas as possible. Formally, we define the ecological serving range as the region within 300 m from a park or an open space, and the ecology metric measures the proportion of residential areas that are covered by the ecological serving range. The metric is calculated as follows:

$$ESR = Union\{Buffer(PA_1, 300), \dots, Buffer(PA_{n_{PA}}, 300), \\ Buffer(OS_1, 300), \dots, Buffer(OS_{n_{OS}}, 300)\}, \quad (11)$$

$$A_{RZ} = \sum_{i=1}^{n_{RZ}} Area(RZ_i), \quad (12)$$



$$A_{RZ}^e = \sum_{i=1}^{n_{RZ}} \text{Area}(\text{Intersection}(RZ_i, ESR)), \quad (13)$$

$$\text{Ecology} = \frac{A_{RZ}^e}{A_{RZ}}, \quad (14)$$

where  $\text{Buffer}(PA_i, 300)$  and  $\text{Buffer}(OS_i, 300)$  represent the regions that extend the park and open space 300 m outward, which is their serving range, and  $ESR$  is the ecological serving range that combines the serving range of all parks and open spaces. The ecology metric encourages the agent to maximize  $A_{RZ}^e$ ; thus, the greenness of the community plan is promoted.

**Traffic.** For the second stage of road planning, we evaluate the traffic efficiency from three perspectives, including density, connectivity and spacing. Road density is the ratio of the total length of roads to the land area. Connectivity is a network characteristic that reflects the strength of how different parts of a network are linked with each other, and we choose the number of connected components and the number of dead-end roads. To achieve appropriate road spacing, we also include two terms to penalize too large (>600 m) and too small (<100 m) spacing. Formally, the traffic metric is calculated as follows given the road plan  $p_R$  and the converted graph  $g_R$  from the planned road network:

$$T_{\text{density}} = \frac{\text{Length}(p_R)}{A_c}, \quad (15)$$

$$T_{\text{connectivity}} = \frac{1}{NCC(g_R)} + \frac{1}{1 + \sum_{v \in g_R} \mathbb{1}[\text{Degree}(v) = 1]}, \quad (16)$$

$$T_{\text{spacing}} = \frac{1}{1 + \sum_{r \in p_R} \mathbb{1}[\text{Length}(v) > 600]} + \frac{1}{1 + \sum_{r \in p_R} \mathbb{1}[\text{Length}(v) < 100]}, \quad (17)$$

$$\text{Traffic} = \frac{1}{3} * (T_{\text{density}} + T_{\text{connectivity}} + T_{\text{spacing}}), \quad (18)$$

where  $\text{Length}$  calculates the length of a road segment,  $A_c$  is the area of the community,  $NCC$  calculates the number of connected components in a network and  $\text{Degree}$  calculates the degree of a node in the graph. Combining the three perspectives, the traffic metric encourages the agent to plan denser roads and, at the same time, guarantees connectivity and appropriate spacing, without creating dead-end roads or planning too-long or too-short road segments.

**Model training.** We train our model for hundreds of iterations to learn the skills of spatial planning. In each iteration, we collect training samples of a few thousand episodes and update the parameters of our model using proximal policy optimization. Specifically, the loss function is a combination of policy loss, policy entropy and value loss. Policy loss is a surrogate clipped objective to improve the policy with safe exploration, which is calculated as follows:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \quad (19)$$

$$L_{\text{policy}} = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t), \quad (20)$$

$$\hat{A}_t = Q(s_t, a_t) - V(s_t), \quad (21)$$

where  $\theta$  is the parameters of our model,  $r_t(\theta)$  is the ratio of the probability of the new policy to the old policy,  $\hat{A}_t$  is the advantage function and clip restricts the update to be not too large. The entropy loss controls the

balance between exploitation and exploration, which is calculated as follows:

$$L_{entropy} = Entropy[Prob(a_1), \dots, Prob(a_{n_a})], \quad (22)$$

where  $n_a$  is the total number of actions that equals to  $M$ (edges) or  $N$  (nodes) in different planning stages, and Prob is obtained by policy networks according to equations (12) and (14). We use mean squared error loss to supervise the value prediction:

$$L_{value} = MSE(\hat{v}_t, R_t), \quad (23)$$

where  $R_t$  is the return value from ground-truth and  $\hat{v}_t$  is estimated by value network according to equation (15). The final loss function is a weighted sum of the above three terms:

$$L = L_{policy} + \beta L_{entropy} + \gamma L_{value}, \quad (24)$$

where  $\beta$  and  $\gamma$  are hyper-parameters in our model.

**Model inference.** After we obtain a well-trained model, we perform model inference to generate community plans. We use the policy networks and compute the probability distribution over different actions according to equations (12) and (14), that is, the probability of selecting different edges and nodes. Then the most likely action is chosen to place land use or road at the location specified by the action:

$$a = \operatorname{argmax}\{Prob(a_1), \dots, Prob(a_{n_a})\}, \quad (25)$$

where  $n_a$  is  $M$  or  $N$  for land use and road planning, respectively. It is worth noting that we can directly perform model inference under a different setup without re-training, and the results are illustrated in Fig. 1c,d.

**Integration with manually designed planning concepts.** The DRL framework is not designed for replacing human designers but serves as an intelligent assistant to improve the productivity of human designers. Specifically, AI models are good at optimizing spatial efficiency in large solution spaces, whereas human designers are good at conceptual prototyping. Therefore, we design a new workflow, in which human and AI collaboratively accomplish urban-planning tasks and leverage their respective expertise. As shown in Supplementary Fig. 7a, we propose a workflow with four key steps of conceptualization, planning, adjustment and evaluation, where AI takes responsibility for the planning step. In the workflow, human designers can leave the heavy and specific planning work to the AI, and they only need to provide relatively abstract conceptual planning and make adjustments to the spatial plans generated by the AI. We represent planning concepts as two major types, center and axis, and each concept is related to one or several land-use functions. For example, in the left part of Supplementary Fig. 7b, the RE center in the HLG community represents the concept that encourages recreation zones near the specified location. Similarly, in the right part of Supplementary Fig. 7b, the BU&OF axis in the DHM community represents the concept that expects a business and office core along the specified band region. We feed the initial conditions of the community and the planning concepts to the model, and then train our DRL model to realize the planning concepts while at the same time optimizing spatial efficiency. As the concept of center and axis is essentially the spatial relationship between specific land-use functions and predefined locations, it can be easily integrated into our framework. Specifically, we utilize customized reward functions to implement planning concepts, that is, we add a reward to reflect the extent of consistency with the planning concept. For the center concept, we calculate the fraction of concept-related land-use functions in the region near the specified center location as

follows:

$$r_c = \frac{1}{n_c} \sum_{j=1}^{n_c} \mathbb{1}[T_j \in T_c], \quad (26)$$

where  $L_a$  is the length of the axis,  $L_a^p$  is the distance of projected points of concept-related land-use functions on the axis,  $n_a$  is the number of land use blocks within 100m from the predefined axis and  $T_a$  is the land-use function related to the concept. This reward encourages the DRL agent to place concept-related land-use functions as evenly as possible in the band area around the axis. The concept reward is combined with efficiency reward, including service and ecology, by weighted sum. Through jointly optimizing efficiency and concept rewards, the DRL agent learns to improve spatial efficiency on the basis of realizing predefined planning concepts.

## 译文 1

作者: Yu Zheng, Yuming Lin, Liang Zhao, Tinghai Wu, Depeng Jin, Yong Li

出处: Nature Computational Science volume 3, pages 748–762 (2023)

### 基于深度强化学习的城市社区空间规划

**摘要:** 有效的城市社区空间规划在城市可持续发展中起着至关重要的作用。尽管地理信息系统和计算机辅助设计带来了便利,但确定土地利用和道路布局仍然很大程度上依赖于人类专家。在这里,我们提出了一种人工智能城市规划模型,用于生成城市社区的空间规划。为了克服城市地理的多样性和不规则性,我们构建了一个图来描述城市的任意形式的拓扑结构,并将城市规划表述为在图上进行顺序决策的问题。为了应对庞大解空间的挑战,我们开发了一种基于图神经网络的强化学习模型。对合成和真实社区的实验表明,我们的计算模型在客观指标上优于人类专家设计的规划,并且可以生成适应不同情境和需求的空间规划。我们还提出了一种人工智能与人类协同工作的城市规划工作流程,在这个过程中,人类设计师可以从我们的模型中受益匪浅,提高工作效率,用更少的时间生成更高效的空间规划。我们的方法展示了计算城市规划的巨大潜力,并为在城市科学中利用计算方法解决具有挑战性的现实问题提供了更多探索的途径。

## 1. 方法

### 1.1 问题建模

我们将社区空间规划问题建模为一个序列的马尔可夫决策过程 (MDP),这是规划智能体与环境之间的交互过程。在这个过程中,智能体观察“状态”(社区的当前状况),在每一步采取“动作”(放置城市功能),并接收环境发出的“奖励”(计划结果的影响),环境根据智能体的动作进行“转换”(布局的变化)。我们利用深度强化学习 (DRL) 来学习一个有效的策略,将状态映射到动作,使用参数化的神经网络。神经网络通过在 MDP 下进行大量训练进行空间效率的优化,训练样本包括数百万个 4 元组 (状态、动作、奖励和转换)。如图 1 所示,我们的 MDP 由两个连续的阶段组成:

- 土地利用规划。在给定初始道路条件的情况下,智能体逐一放置功能块,可以是靠近现有道路或靠近先前放置的土地利用边界。在分配所有功能和开放空间之后,将关于土地利用效率的奖励返回给智能体,智能体将不同的土地利用视为一个整体系统。最终的土地利用计划成为道路规划的初始条件。

- 道路规划。已规划土地利用的边界是道路建设的可行位置。智能体迭代地建造道路，将一个边界转化为一个道路段。在预定义的终止步骤处停止时，将考虑交通效率的奖励返回给智能体。

奖励仅在每个阶段的最后一步进行计算，以总结土地利用规划和道路规划的性能，各自的所有中间步骤都获得奖励为 0。我们基于 15 分钟城市的概念，强调空间效率以促进步行和骑行等积极交通，而不是依赖汽车。这两个奖励项的计算如下：

$$r_L = \alpha Service + Ecology, \quad (1)$$

$$r_R = Traffic, \quad (2)$$

其中，*Service* 衡量了在 15 分钟城市中社区生活循环指数，*Ecology* 衡量了绿地和公园的覆盖率，*Traffic* 是道路密度和连通性的组合（详见方法部分）， $\alpha$  是一个超参数，表示服务性能在土地利用奖励中的权重。通过计算的奖励值，我们使用近端策略优化来更新值网络和策略网络的参数。我们首先对智能体进行土地利用规划任务的训练，直到收敛，然后使用在第一阶段获得的最优土地利用计划对智能体进行道路建设的训练。经过两个阶段的训练，AI 智能体能够设计出具有高效空间布局的社区，包括土地利用和道路。

**图模型。**与先前的步行和区域设计任务不同，城市规划更具挑战性，因为它在问题形式上具有更大的自由度。具体而言，先前任务的条件是规则的，例如在一个  $19 \times 19$  的棋盘上放置石头或将矩形宏单元放置在网格芯片画布上，这可以用像素（栅格）表示。相比之下，社区空间规划的条件是多样且不规则的，因为道路的拐角和土地块通常不是正交的。为了准确描述城市地理元素，包括土地块（L）、道路和土地利用边界的段（S）以及道路和土地利用边界之间的交叉口（J），我们使用向量表示。在城市规划中，与栅格表示相比，向量表示已被证明具有显著优势，并由以下三个几何元素组成：

- ‘Polygon’，描述待规划的空地（例如，图 2a 右侧的 L1）或已规划的土地块（例如，图 2a 右侧的 L2），其包含土地边界的坐标；
- ‘LineString’，表示道路段（例如，图 2a 右侧的 S3）或土地块的边界边缘（例如，图 2a 右侧的 S9），其包含起点和终点的坐标；以及
- ‘Point’，代表道路和土地块边界之间的交叉口（例如，图 2a 右侧的 J2 和 J7），具有它们的坐标。

我们将所有地理元素转换为上述三类几何形状，然后将整个社区表示为一个图，其中节点是几何形状，边表示这些几何形状之间的空间相邻关系，即如果底层的两个几何

形状相互接触，则连接两个节点。每个节点将其地理信息存储为节点特征，包括几何形状的类型、坐标、宽度、高度、长度和面积。通过这种方式，空间规划可以被转化为在动态图上进行选择的问题（图 2），其中图根据智能体的动作演变。

在土地利用规划任务中，智能体选择连接空地和交叉口的一个 L-J 边，将给定的功能放置在相应 L 和 J 指定的位置（图 2a）。在每一步中，由于新放置的功能会生成新的节点和边，因此相邻图的拓扑结构会发生变化。新节点包括新功能本身、其边界、新的交叉口和分割段。新的边表示新建立的空间相邻关系。类似地，在道路规划任务中，智能体选择当前是边界的一个 S 节点，并将其构建为一个道路段（图 2b）。尽管拓扑结构保持不变，但由于所选节点的类型从边界变为道路，图的属性发生变化。通过使用图模型对问题进行重新构建，我们现在可以处理不规则的城市块，并在一个单一的图上统一地处理土地利用和道路规划这两个看似不同的阶段。

**行动空间设计。**城市规划的另一个主要挑战是庞大的行动空间，原始连续空间中几乎是无限的，在减小的离散图空间中仍然过大。随着每一步放置一个功能，相邻图将不断增长，导致一个具有数千个节点和边的大型图。一个  $2\text{ km} \times 2\text{ km}$  社区的典型空间规划在每个阶段可能需要 100 个规划步骤，相邻图可能具有 4,000 个边和 1,000 个节点，使得两个阶段的行动空间分别为  $4,000^{100}$  和  $1,000^{100}$ 。此外，在空间中，有效的行动非常稀疏，而且大部分行动的质量较低，可能导致不合理的结果，比如在空地中央放置设施而不连接道路。因此，减小行动空间并避免不合理的行动至关重要。

为了解决这一挑战，我们提出了一个通用的深度强化学习（DRL）框架，在这个框架中，智能智能体在一个减小的图空间中感知并做决策，而环境则处理原始地理空间中的城市元素，并根据地理空间布局生成图状态。与此同时，我们将整个行动空间分解为三个子空间的笛卡尔积，包括要规划什么、在哪里规划以及如何规划，让 DRL 智能体专注于核心问题——在哪里规划。通过领域知识，可以通过固定不同土地利用类型的规划顺序来消除第一个要规划的子空间，允许更依赖于初始道路网络的土地利用类型较早规划（详见方法）。为了避免明显不当的规划行动，在哪里规划上，我们对智能体的行动施加规划约束，使用一个行动掩码阻止不合理的选项，即在两个规划阶段，只有 L-J 边和 S 节点是候选的。在选择给定土地利用类型的一个 L-J 边之后，功能被放置在相应的土地块（L 节点）上，位于相应交叉口（J 节点）的位置，其形状和大小由预定义的规则确定，以最大化对现有道路和边界的重用（详见方法）；因此，有效地消除了如何规划的最后一个子空间。通过这些设计，我们将行动空间缩小到可解决的规模，并过滤掉大多数不合理的行动，为 DRL 算法提供了有效的优化。总之，空间规划的原始问题成功地转化为具有适度行动空间的标准序列决策过程。

**框架。**在进行上述问题重构和行动空间设计后，我们提出了一个深度强化学习（DRL）框架，其中一个 AI 智能体通过与空间规划环境的交互学习布局土地利用和道路，如图 3 所示。序列 MDP（图 3e, f）包含以下关键组件：

- 状态（States）总结了包含丰富节点特征的先前引入的相邻图的当前空间计划，以及其他信息，例如不同土地利用类型的统计数据。
- 动作（Actions）指示放置当前土地利用或构建新道路段的位置，这些位置是从相邻图中选择的边缘或节点转换而来的。
- 奖励（Rewards）在所有中间步骤为 0，除了每个阶段的最后一步，其中它评估土地利用和道路的空间效率。
- 转移（Transitions）描述了在给定所选位置的情况下布局的变化，转移发生在原始地理空间（地图上的新土地利用和道路）和转换后的图空间（图的新拓扑和属性）中。

在每一步中，智能体通过使用图神经网络（GNN）对图进行编码，表示状态。通过多次消息传递和非线性激活层，GNN 状态编码器生成边、节点和整个图的有效表示（图 3a），这将被值网络和策略网络利用（图 3b-d）。具体而言，由于为土地利用选择位置等同于在图上选择边缘，土地利用策略网络使用边缘嵌入，并使用边缘排序 MLP 为每个边缘评分，如图 3b 所示。对于每个边缘，获得的分数表示相应边缘的采样概率，该概率返回给环境，并成为在该边缘指定的位置放置土地利用的概率。类似地，在道路规划中，道路策略网络使用节点嵌入，并使用节点排序 MLP 为每个节点评分（图 3d），输出选择一个土地块边界并将其构建为道路段的概率。最后，值网络接收总结整个社区的图嵌入，并使用全连接层预测规划奖励（图 3c）。为了掌握空间规划的技能，在训练过程中，由提出的模型完成数百万个空间计划，以在大规模解空间中进行搜索，并用作实时训练数据来更新神经网络的参数。

## 1.2 详细方法

**框架。**如论文中介绍的，我们使用矢量几何，包括多边形（Polygon）、线串（LineString）和点（Point）来描述城市地理元素。具体而言，有十种土地块类型被表示为多边形，包括待规划的初始空地，以及九种不同的功能类型，分别是住宅（RZ）、学校（SC）、医院（HO）、诊所（CL）、商业（BU）、办公室（OF）、娱乐（RE）、公园（PA）和开放空地（OP）。此外，有两种由线串表示的段（道路和土地利用边界）和一种由点表示的交叉口（道路和土地利用边界之间的交叉点）。因此，一个社区通过几何形状表格忠实地

表示，其中每一行都是一个带有 ID、类型和几何形状三列的地理元素。社区的初始条件包括所有原始土地块、道路和交叉口，其准确坐标由几何形状表格中的相应几何形状记录。在合成的网格社区中，我们对一个尺寸为  $2.4\text{ km} \times 2.4\text{ km}$  的基本社区进行实验，其中包含 16 个矩形空地、40 个水平或垂直的初始道路段和 25 个道路交叉口，如图 1 的第一步所示。在真实的社区中，我们使用 OSMnx30 和 geopandas 从 OpenStreetMap 复制了北京的 HLG 和 DHM 社区的道路网络，保留住宅区块，并将其他区域留作待整修的空地。最终，我们获得了两个约为  $4\text{ km}^2$  的社区，如图 1a 和图 11a 所示。

**规划需求和要求。**在进行实际空间规划之前，我们需要确定规划的需求和要求，这充当规划环境的配置。规划需求描述了每种土地利用类型必须达到的数量，可以是面积或数量，例如，住宅区块占社区面积的 50% 和三个医院。与此同时，我们还对每个规划块的最小面积（以平方米为单位）有要求；例如，一个学校的面积至少为  $10,000\text{ m}^2$ 。表 3 显示了一个社区规划需求 and 要求的示例，其中社区面积的 15% 需要规划为公园；因此，它作为一个绿色社区。仅考虑满足所有需求 and 要求的空间规划作为成功的实例，并将其保留为训练样本，而失败的实例则被丢弃。在我们的框架中，规划需求 and 要求是环境的配置，使得我们的模型在生成空间规划时非常灵活。具体而言，一旦我们在一个配置下获得了一个训练良好的模型，我们可以简单地更改配置，并直接进行模型推断而无需重新训练，以生成满足不同规划需求 and 要求的计划，例如图 1c、d 中不同服务供应的社区规划。

**土地利用类型的规划顺序。**正如论文中介绍的，为了减小庞大的行动空间，我们基于领域知识固定了不同土地利用类型的规划顺序，并使智能体专注于选择位置的核心任务。由于可行的位置都在现有的交叉口附近，先规划的土地利用将更靠近初始道路，交通更为便利。因此，我们首先规划更依赖道路的设施，包括医院（诊所）、学校和娱乐设施。同时，在土地利用规划的后期步骤，可行的空地形状趋向更加不规则和分散，这对通常占据整个地块的住宅区块不太适合；因此，在规划了以上依赖道路的设施后，我们分配住宅区块。最后，我们安排那些在土地形状上要求不高的土地利用类型。在满足所有规划需求之后，剩余的空地被分配为开放空地。总之，在我们的框架中，规划顺序固定为：医院、学校、诊所、娱乐、住宅、公园、办公室、商业和开放空地。让智能体确定土地利用类型的顺序可能是一种替代方法。然而，这会使问题变得更加复杂，因为行动空间会急剧增加。在实践中，我们固定的顺序生成了合理的空间规划。

**土地切割。**在土地利用规划中，环境接收来自智能体的动作，即所选的 L-J 边，然后在相应的土地块（L 节点）上在相应交叉口（J 节点）的位置切割出一块新的土地。我们开发了一个基于规则的系统，其中融入了专家知识，以确定新土地的形状和大小。基于



规则的系统大致由三个步骤组成：（1）确定 J 和 L 之间的关系，比如在道路中间或在角落。（2）确定沿着从交叉口 J 延伸的现有边界的参考线，可以是 I 形、L 形和 U 形。（3）确定从参考线向块 L 内部延伸的长度，形成最终切割的新土地。这三个步骤根据专家知识进行，以满足规划要求并尽可能贴近当前规划。

**状态。**我们的状态包含三个部分：（1）城市相邻图，（2）当前要放置的对象和（3）社区统计。我们构建一个图来表示当前社区信息，如图 2 所示，其中节点是城市地理元素，边表示空间相邻关系。我们计算丰富的地理属性作为节点特征，包括基础城市元素的类型、坐标、面积、长度、宽度和高度。边由稀疏邻接矩阵表示。至于当前要放置的对象，其类型由环境根据规划需求和规划顺序确定，即，环境将遍历规划顺序，并在前一类型的规划需求满足后转移到下一类型。我们将当前对象视为一个虚拟孤立节点，其类型特征由环境提供，其他节点特征保留为默认值。最后，社区统计包括当前计划中不同土地利用类型的面积和数量，以及规划需求，总结了当前空间规划的状况和进展。

**动作。**如图 2a 所示，土地利用规划被重新定义为一个序列 MDP，智能体在动态图中选择一条边。因此，土地利用规划的动作空间是从 N 条边中选择的概率分布，我们从这个分布中采样以获得动作。类似地，道路规划是一个序列 MDP，如图 2b 所示，智能体选择节点；因此，道路规划的动作空间是对 M 个节点的概率分布，从中采样以生成节点选择动作。此外，如前面介绍的，我们对动作空间施加了约束；例如，智能体只能选择 L-J 边（连接空地和交叉口）和 S 节点（土地利用边界），以避免不合理的空间规划。因此，我们在每一步计算一个掩码，指示可行的选项，概率分布将乘以该掩码，只允许可行的边或节点作为采样动作。

**策略和价值网络。**如图 3b-d 所示，我们开发了独立的策略网络，分别在策略和价值阶段采取行动，以及一个价值网络来预测空间规划的性能。这三个网络共享相同的状态编码器，以获取状态表示，充分利用图神经网络（GNN）。策略网络通过对图中的边和节点进行评分生成概率分布，然后从该分布中采样以采取行动。与此同时，价值网络评估整个图以预测空间效率，为社区规划提供反馈。在本节中，我们介绍这三个网络的详细设计。

**土地利用策略网络。**在土地利用规划中，智能体将当前对象放置在由选定边指定的位置。边的选择效果与边和当前对象都有关系；例如，在已规划医院旁边放置另一家医院可能导致服务效率较低。因此，土地利用策略网络将边和当前对象都视为输入。如图 3b 所示，我们开发了一个前馈网络，即边排序的 MLP，用于评分每条边：

$$s(e_{ij}) = FF_{land}(e_{ij}^L || v_c || e_{ij}^L - v_c || e_{ij}^L \cdot v_c), \quad (3)$$

其中  $e_{ij}^L$  和  $v_c$  的差异和内积也被连接起来，以强调当前要规划的对象与已经规划的土地利用之间的关系。分数通过 softmax 转换为所有边上的概率分布：

$$Prob(e_{ij}) = \frac{e^{S(e_{ij})}}{\sum_{s,t \in E} e^{S(e_{st})}}, \quad (4)$$

其中通过采样选择一条边。

**道路策略网络。**在道路规划中，智能体选择一个边界节点并在其位置规划一条道路。与土地利用规划不同，图的拓扑结构是稳定的，没有新节点需要添加。因此，无需包含当前对象。同时，道路策略网络以节点嵌入作为输入，这些嵌入已经通过 GNN 的消息传递包含了邻近的地理信息。如图 3d 所示，采用另一个节点排序的前馈 MLP 来评分每个节点：

$$s(v_i) = FF_{road}(v_i). \quad (5)$$

分数也通过 softmax 运算符转换为概率：

$$Prob(v_{ij}) = \frac{e^{S(v_{ij})}}{\sum_{j \in N} e^{S(v_j)}}, \quad (6)$$

然后从该概率分布中采样以选择一个节点。

**价值网络。**如图 3c 所示，我们开发了一个价值网络来评判当前的规划状况并预测规划的性能。因为这是对整个社区的整体评估，我们将图级表示作为价值网络的输入。同时，我们还包括社区统计信息。具体来说，我们将图表示和统计信息嵌入连接在一起，并采用全连接层来预测性能：

$$\hat{v} = fa(g^L || h_s), \quad (7)$$

其中  $\hat{v}$  是当前计划的估计值。

**奖励。**我们训练策略网络以优化空间布局的效率，涉及到服务、生态和交通。正如本文的方程 (1) 和 (2) 所示，我们定义了奖励函数，对上述指标进行综合评估。同时，奖励值可以在几十毫秒内快速计算给定一个空间计划，从而有可能收集大规模的样本来训练 DRL 模型。在本节中，我们介绍了如何计算这三个指标。值得注意的是，我们的框架是灵活的，可以扩展以包含更多的奖励指标。

**服务。**我们采用了 15 分钟生活圈的概念，该概念要求社区的基本服务在 15 分钟内可步行或骑自行车到达。具体而言，如图 1b、c 所示，我们考虑了五种不同的基本服务，每种服务与一个或两个设施相关，即教育（学校）、医疗（医院、诊所）、工作（办公室）、购物（商业）和娱乐（休闲）。因此，15 分钟生活圈意味着设施和居住区之间的距离需

要小于 15 分钟的步行距离，我们的实验中设置为 500 米。我们将服务度量定义为 500 米范围内可达服务的比例，并将该度量值平均到所有居住区。形式上，给定社区的空间规划  $p$ ，服务度量的计算如下：

$$d(i, j) = \min\{EucDis(RZ_i, FA_1^j), \dots, EucDis(RZ_i, FA_{n_j}^j)\}, \quad (8)$$

$$Service_i = \frac{1}{5} \sum_{j=1}^5 \mathbb{1}[d(i, j) < 500], \quad (9)$$

$$Service = \frac{1}{n_{RZ}} \sum_{i=1}^{n_{RZ}} Service_i, \quad (10)$$

其中  $EucDis$  是欧几里得距离， $d(i, j)$  是第  $i$  个居住区  $RZ_i$  访问由设施  $FA^j$  提供的第  $j$  个服务的最小距离， $n_j$  是设施  $FA^j$  的总数。 $Service_i$  是第  $i$  个居住区的 15 分钟生活圈度量，我们对所有  $n_{RZ}$  个居住区进行平均，以获得整个社区的最终服务度量。这个服务度量引导智能体以更为分散的方式安排设施，并使其靠近居住区，这对于增强社区服务的能力至关重要。

**生态。**社区的生态环境对居民的身体和心理健康至关重要；因此，我们包括了一个生态度量，用于衡量公园和开放空间的布局效率。一般而言，公园和开放空间为居住在附近的居民提供服务，我们希望它们能为尽可能多的居住区提供服务。形式上，我们将生态服务范围定义为距离公园或开放空间 300 米内的区域，生态度量衡量了被生态服务范围覆盖的居住区的比例。度量的计算如下：

$$ESR = Union\{Buffer(PA_1, 300), \dots, Buffer(PA_{n_{PA}}, 300), \\ Buffer(OS_1, 300), \dots, Buffer(OS_{n_{OS}}, 300)\}, \quad (11)$$

$$A_{RZ} = \sum_{i=1}^{n_{RZ}} Area(RZ_i), \quad (12)$$

$$A_{RZ}^e = \sum_{i=1}^{n_{RZ}} Area(Intersection(RZ_i, ESR)), \quad (13)$$

$$Ecology = \frac{A_{RZ}^e}{A_{RZ}}, \quad (14)$$

其中  $Buffer(PA_i, 300)$  和  $Buffer(OS_i, 300)$  表示将公园和开放空间向外延伸 300 米的区域，这是它们的服务范围， $ESR$  是所有公园和开放空间服务范围的组合。生态度量鼓励智能体最大化  $A_{RZ}^e$ ；因此，社区规划的绿化程度得到提升。

**交通。**对于道路规划的第二阶段，我们从三个角度评估交通效率，包括密度、连接

性和间距。道路密度是指道路总长度与土地面积的比率。连接性是反映网络不同部分如何相互连接的网络特性，我们选择连接组件的数量和死胡同道路的数量。为了实现适当的道路间距，我们还包括两个项目，对过大的（>600 米）和过小的（<100 米）间距进行惩罚。形式上，给定道路规划  $p_R$  和从规划的道路网络转换而来的图  $g_R$ ，交通度量计算如下：

$$T_{density} = \frac{Length(p_R)}{A_c}, \quad (15)$$

$$T_{connectivity} = \frac{1}{NCC(g_R)} + \frac{1}{1 + \sum_{v \in g_R} \mathbb{1}[Degree(v) = 1]}, \quad (16)$$

$$T_{spacing} = \frac{1}{1 + \sum_{r \in p_R} \mathbb{1}[Length(v) > 600]} + \frac{1}{1 + \sum_{r \in p_R} \mathbb{1}[Length(v) < 100]}, \quad (17)$$

$$Traffic = \frac{1}{3} * (T_{density} + T_{connectivity} + T_{spacing}), \quad (18)$$

其中  $Length$  计算道路段的长度， $A_c$  是社区的面积， $NCC$  计算网络中连接组件的数量， $Degree$  计算图中节点的度数。结合这三个角度，交通度量鼓励智能体规划更密集的道路，同时确保连接性和适当的间距，避免产生死胡同道路或规划过长或过短的道路段。

**模型训练。**我们对模型进行数百次迭代的训练，以学习空间规划的技能。在每次迭代中，我们收集几千个情节的训练样本，并使用近端策略优化来更新我们模型的参数。具体而言，损失函数是策略损失、策略熵和值损失的组合。策略损失是一个替代的被修剪的目标，通过安全的探索来改善策略，计算如下：

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \quad (19)$$

$$L_{policy} = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t), \quad (20)$$

$$\hat{A}_t = Q(s_t, a_t) - V(s_t), \quad (21)$$

其中  $\theta$  是我们模型的参数， $r_t(\theta)$  是新策略概率与旧策略概率的比率， $\hat{A}_t$  是优势函数， $\text{clip}$  将更新限制在不太大的范围内。熵损失控制开发和探索之间的平衡，计算如下：

$$L_{entropy} = \text{Entropy}[\text{Prob}(a_1), \dots, \text{Prob}(a_{n_a})], \quad (22)$$

其中  $n_a$  是不同规划阶段中等于  $M$ （边缘）或  $N$ （节点）的总动作数，而  $\text{Prob}$  是根据方程（12）和（14）由策略网络获得的。我们使用均方误差损失来监督值的预测：

$$L_{value} = \text{MSE}(\hat{v}_t, R_t), \quad (23)$$

其中  $R_t$  是来自地面实况的回报值,  $\hat{v}_t$  是根据方程 (15) 由值网络估计的。最终的损失函数是上述三个项的加权和:

$$L = L_{policy} + \beta L_{entropy} + \gamma L_{value}, \quad (24)$$

其中  $\beta$  和  $\gamma$  是我们模型中的超参数。

**模型推断。** 在我们获得一个训练良好的模型之后, 我们进行模型推断以生成社区规划。我们使用策略网络, 根据方程 (12) 和 (14) 计算不同动作的概率分布, 即选择不同边缘和节点的概率。然后选择最有可能的动作在指定的位置放置用地或道路:

$$a = \operatorname{argmax}\{\operatorname{Prob}(a_1), \dots, \operatorname{Prob}(a_{n_a})\}, \quad (25)$$

其中  $n_a$  对于土地使用规划和道路规划分别为  $M$  或  $N$ 。值得注意的是, 我们可以在不重新训练的情况下直接进行模型推断, 并且结果如图 1c、d 所示。

**与手工设计的规划概念集成。** DRL 框架并非旨在替代人类设计师, 而是作为智能助手, 提高人类设计师的生产力。具体而言, 人工智能模型擅长在大型解决方案空间中优化空间效率, 而人类设计师擅长概念原型设计。因此, 我们设计了一种新的工作流程, 其中人类和人工智能共同完成城市规划任务, 并发挥各自的专业知识。如图 7a 所示, 我们提出了一个包含概念化、规划、调整和评估四个关键步骤的工作流程, 其中人工智能负责规划步骤。在这个工作流程中, 人类设计师可以将繁重和具体的规划工作交给人工智能, 他们只需提供相对抽象的概念规划, 并对人工智能生成的空间规划进行调整。我们将规划概念表示为两种主要类型, 即中心和轴, 每个概念与一个或多个用地功能相关。例如, 在图 7b 的左侧, HLG 社区的 RE 中心表示鼓励在指定位置附近设置休闲区的概念。类似地, 在图 7b 的右侧, DHM 社区的 BU & OF 轴表示期望在指定的带状区域设置商业和办公核心的概念。我们将社区的初始条件和规划概念输入模型, 然后训练我们的 DRL 模型以实现规划概念, 同时优化空间效率。由于中心和轴的概念本质上是特定用地功能与预定义位置之间的空间关系, 因此它可以很容易地集成到我们的框架中。具体而言, 我们使用定制的奖励函数来实现规划概念, 即我们添加奖励以反映与规划概念的一致性程度。对于中心概念, 我们计算靠近指定中心位置的区域内与概念相关的用地功能的比例如下:

$$r_c = \frac{1}{n_c} \sum_{j=1}^{n_c} \mathbb{1}[T_j \in T_c], \quad (26)$$

其中  $L_a$  是轴的长度,  $L_a^p$  是轴上概念相关用地功能的投影点距离,  $n_a$  是距离预定义轴 100m 以内的用地块数量,  $T_a$  是与概念相关的用地功能。此奖励鼓励 DRL 智能体在轴

周围的带状区域均匀地放置与概念相关的用地功能。概念奖励与效率奖励（包括服务和生态）通过加权和组合在一起。通过共同优化效率和概念奖励，DRL 智能体学会在实现预定义规划概念的基础上提高空间效率。

## Puzzle-based parking

Parag J. Siddique, Kevin R. Gue, John S. Usher

Department of Industrial Engineering, University of Louisville, Louisville, KY 40292, USA

**Abstract:** In a common parking lot, much of the space is devoted not to parking but to lanes in which cars travel to and from parking spaces. Lanes must not be blocked for one simple reason: a blocked car might need to leave before the car that blocks it. Self-parking and intelligent communication capabilities of autonomous vehicles introduces an opportunity to overcome this constraint, and therefore to achieve much higher storage capacity of cars. We show how to maximize the number of cars in a parking lot that assumes interfering cars could be moved out of the way by a centralized controller. We provide optimal results for small lots with a single entry point, and we offer heuristic methods for larger lots. Improvements in parking capacity of 80 percent is possible.

### 1 A puzzle-based parking lot

The prospect of autonomous passenger cars presents many interesting opportunities to improve urban transportation. With respect to parking, the potential for remote control of cars would allow, for example, cars to be parked closer together because there would be no need for doors to open to allow passengers to exit. The potential for improvement in parking density is estimated to be 20 percent. Another thought is that autonomous cars do not need to park in a busy urban area. Rather, a remote parking zone can be created in inexpensive suburban areas. Another widely accepted thought is, autonomous vehicles will eliminate private vehicle ownership. Autonomous taxis will bring shared mobility, which will reduce parking demand in a significant amount.

In this context, Bosch , collaborating with Mercedes and Ford, has designed an automated valet parking lot for autonomous vehicles. Upon arrival at the parking lot, human rider leaves the car at a drop-off point. The parking lot is equipped with smart sensor systems which guide vehicles to a parking spot. Whenever the rider needs the car, it can be retrieved using a smartphone app, and the car comes to the pick-up point. It is interesting to note that no human is needed to control the cars, and no additional civil infrastructure is needed to be built.

In such an automated parking lot, it is possible to achieve higher parking density by eliminating driving lanes. In a common parking lot, driving lanes cannot be blocked for one simple reason: a blocked car might need to leave before the car that blocks it. However, an automated valet parking lot creates an opportunity to overcome this problem, and to achieve higher parking density. Therefore, we already have the required technology to achieve higher density, all we need is efficient facility design and planning.

Some authors have envisioned parking schemes to improve the density of parked cars, or equivalently, the capacity of the parking lot. Ferreira et al. were first to propose high density parking for autonomous vehicles. In their design, cars are parked by blocking each other in parallel lanes, similar to the deep lane storage systems used in material handling systems(Fig. 1a). They assume that there is an autonomous parking controller that controls vehicle mobility inside the parking lot. The same group of researchers has extended this work to consider various operational parameters such as parking space per car, number of maneuvers, travel distance, and retrieval time. Their design increases parking capacity by 50 percent.

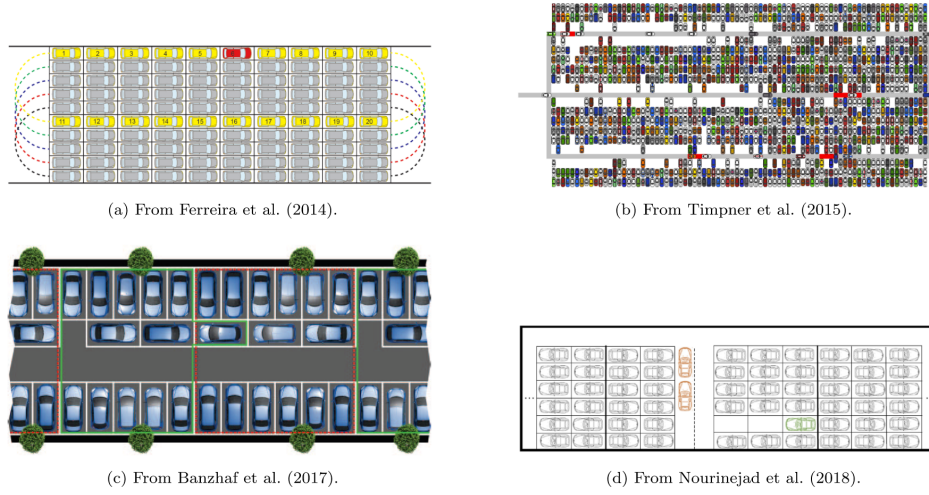


Fig. 1 High density parking designs from other authors. (a) Ferreira et al. (2014) propose a design to park cars by blocking each other in parallel lanes. They allocate a buffer area on both the entrance and the exit of the facility so that cars can make necessary moves for relocations. (b) Timpner et al. place perpendicular aisles between the lanes of the blocked cars to reduce the number of interfering cars. Blocking vehicles relocate to the aisle to clear the path for the leaving vehicle. (c) Banzhaf et al. modify existing parking lots by adding an aisle of blocking cars. (d) In the design of Nourinejad et al., the width of the aisle varies with the depth of the lane to help accommodate relocating cars.

To reduce the number of interfering cars, Timpner et al. modified the design of Ferreira et al., by reducing the lane depth and allowing cars to depart from either end of a lane. The authors estimate density improvement of 33 percent. Banzhaf et al. propose modifying existing layouts by allowing perpendicular lanes of blocking cars within existing aisles. They report an increase in density of 25 percent. Nourinejad et al. describe a similar system in which the width of the aisle varies with the depth of the lane to help accommodate relocating cars. The authors develop a mixed-integer non-linear program to minimize the expected number of vehicle relocations. However, all these researchers are planning for large parking lots which are more appropriate for suburban areas.

From the literature we have learned that an autonomous car will not sit idle in the parking lot. After dropping a passenger, it will be used to transport another passenger. However, due to the gap between supply and demand, there will be some idle time when there is no trip, hence the car must park somewhere. If these cars go to suburban areas for parking, they will not be able to respond to customer calls. Therefore, autonomous vehicles will not prefer to go outside of the city for parking, but instead, it will cruise around the city center to kill time which will lead to city congestion. Therefore, we cannot completely eliminate urban parking. Rather, we can look for ways to make parking easily accessible and cheaper. To do this, we can offer small and dense parking lots for urban areas. Such parking lots can be compared to micro-fulfillment centers in the retail industry. Micro-fulfillment centers are small warehouses located near customers which are designed for on-demand e-commerce fulfillment. Whenever an autonomous vehicle is idle, it can be parked in these small lots. As these lots will be located in urban areas, whenever a car receives a trip request it can serve the request quickly. As a valued-added service: vehicle charging, cleaning or similar services can also be introduced in such parking lots.

In this paper, we study high-density layout design of a parking lot for autonomous vehicles. Our work differs from existing literature in two important ways:

1. We develop optimal designs for small parking lots, which one might expect to find in urban settings.



2. All existing research presupposes a layout and then investigates optimal parameters and performance with respect to that layout. We ask and answer the more basic question of how many cars can be parked in a lot, without assuming any particular layout or structure. As we will show, this relaxation leads to designs with maximum density without the traditional “row-and-aisle” parking structure.



Fig. 2 The Rush Hour puzzle. The goal of the game is to move the red car to the exit directly on the opposite side by moving other cars out of the way.

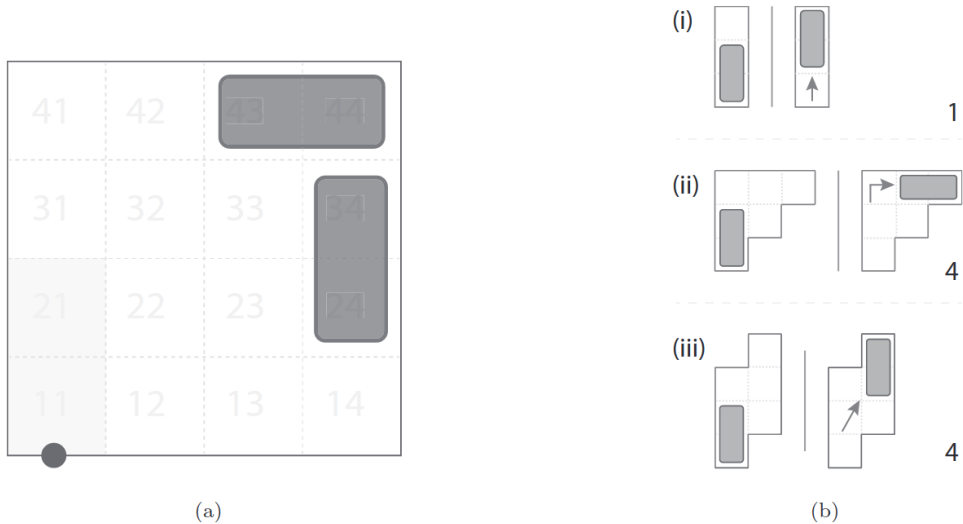


Fig. 3 (a) An example of a  $4 \times 4$  Parking lot with two cars, one car is parked horizontally on cells (43, 44), and another is parked vertically on cells (23, 34), and cells (11, 21) are the I/O point. (b) Three valid moves inside a parking lot. Each of these moves is assigned a weight according to number of cells travelled.

The problem we address is similar to the Rush Hour puzzle (Fig. 2), which is the subject of a considerable literature in the theoretical computer science community. Flake and Baum showed that the generalized version of Rush Hour with an  $n \times n$  grid is PSPACEcomplete. Tromp and Cilibrasi and Hearn and Demaine showed

that Rush Hour with cars only (no  $3 \times 1$  trucks) is also PSPACE-complete. Our problem is different from Rush Hour in a number of ways: the game allows only forward and backward movement of vehicles, whereas real cars can turn; and, the objective of Rush Hour is to remove one specific car, whereas a real parking lot must allow arrival or departure of any vehicle requesting it. High-density storage without defined aisles has also been addressed in the material handling literature, where unit-loads or containers are stored and retrieved using conveyor modules which can move in the four cardinal directions. Whereas in our parking lot problem, cars move on its own without help of any conveyor module, and cars can travel in different directions.

## 2 Modeling a puzzle-based parking lot

We consider a parking lot as a rectangular grid. Cars can be parked horizontally or vertically. We assume cars take two cells in the grid, which means the car size is  $1 \times 2$ . There is one entrance/exit point in the parking lot on the lower-left corner. We call this point as input–output (I/O) point. In Fig. 3a, two cars are parked in a  $4 \times 4$  parking lot, one in cells (43, 44) and the other in cells (24, 34). Cells (11, 21) represent the I/O point.

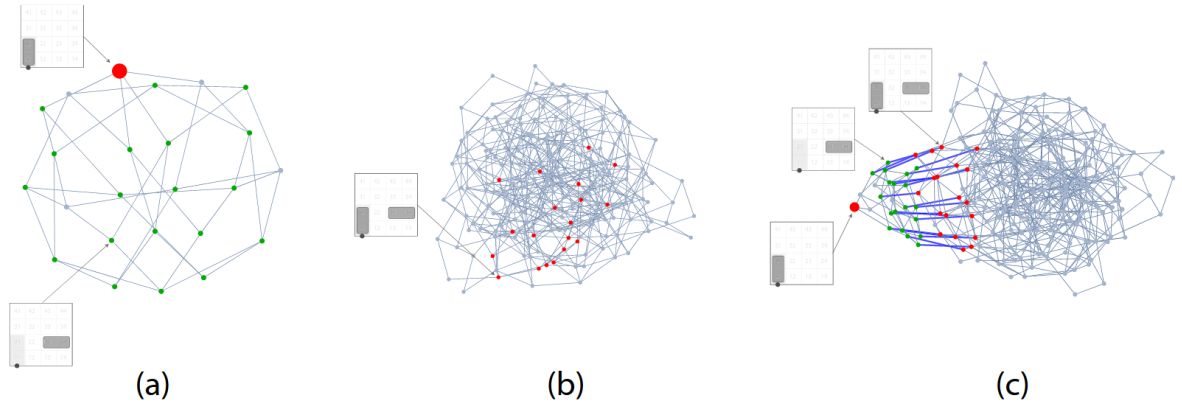


Fig. 4 Process to generate the state space graph of a  $4 \times 4$  parking lot. (a) The 1-car graph. The large red node is the root node—the very first node in this graph. The green nodes are open nodes. Open nodes always allow a new car to enter the parking lot. The root node and one of the open nodes are annotated. (b) 2-car graph in a  $4 \times 4$  grid. Red colored nodes are the initial nodes which include the I/O cells. One of the initial nodes is annotated. (c) Bridging 1-car graph and 2 car graph. Open nodes from the 1 car graph connect the initial nodes from the 2-car graph, highlighted in blue. These connectors are called bridging edges. Position of earlier annotated nodes are also shown.

Cars can make three types of moves inside the lot: straight, right angle, and parallel, and they accomplish these maneuvers in forward or reverse (Fig. 3b). Assuming cars take one unit of time to travel one cell, each movement can be assigned a weight to quantify travel time. For instance, vehicles travel one cell when making a straight move, so this move is assigned a weight of one. Similarly, vehicles travel four cells when making right angle and parallel moves, so these moves are assigned a weight of four. We assume there is a central controller agent in the parking lot, that can communicate with the vehicles to execute all the movements.

Readers may note here that we have not specified dimension of a cell. The size of a cell is decided based on vehicle dimension and kinematics. Rather, we want to leave this cell design for the subject matter experts.

### 2.1 How many cars can be parked in a parking lot?

We model cars in a parking lot with a state space graph  $G = (V, E)$ , where  $V$  is the set of vertices representing configurations of parking lot, and  $E$  is the set of edges representing feasible transitions between configurations. A vertex is also synonymously referred as node in this paper. A configuration is defined as the size of a parking lot, number of cars parked and their locations. The term state is also used as an alternative to

configuration. We want to know: how many cars can be parked in a parking lot? We solve a small example, a  $4 \times 4$  parking lot, to get started with this problem.

In a  $4 \times 4$  parking lot, there are sixteen cells, and each car takes two cells. Therefore, we can place eight cars at most. We begin by generating the state space graph for a  $4 \times 4$  grid. Without considering maneuverability, a single car can be placed in 24 distinct locations, each representing a potential node in the graph. We then execute a pair-wise search of all potential nodes to determine whether they can be connected by a feasible maneuver. We refer to the result as the 1-car graph (Fig. 4a). Next, we place two cars in every possible way to create the 2-car graph. We then eliminate configurations with overlapping cars. For example, configuration  $[(11, 21), (21, 22)]$  is infeasible because cell 21 is used by both of the cars, which is not physically possible. We also eliminate configurations that occupy an entire row or column because they are infeasible (except  $[(11, 21), (31, 41)]$ ). Finally, we connect the nodes using their respective moves to construct the complete 2-car graph (Fig. 4c). Similarly, we can continue to construct graphs up to eight cars.

We find an interesting feature to connect these eight graphs. In an empty parking lot, when the first car arrives, it is initially parked on the I/O cells. This is the starting node of the 1-car graph. This car can move to different locations inside the parking lot. Once the car leaves the I/O cells, the parking lot is open to receive a second car. When the second car arrives, it parks itself on the I/O cells. This is the starting point of the 2 car graph. After that, cars can be relocated to other locations in the parking lot. Similarly, when both of these cars are parked anywhere except the I/O cells, the parking lot is open to receive a third car, and we can continue the process up to 8 cars. Therefore, we observe that each of the  $k$  car graphs, ( $k = 1, 2, \dots, 8$ ) starts from the I/O cells. Hence, we define a parking lot configuration that includes occupied I/O cells as initial node. Initial nodes of the 2-car graph are highlighted in red in Fig. 4b. Note that each  $k$  car graph has multiple initial nodes, except the 1-car graph. To distinguish this node from other initial nodes, we define it as the root node (see large red node in the Fig. 4a). We also observe that to accommodate a new car into a parking lot, we have to make sure that no car is parked on the I/O cells. Such a configuration—I/O cells are unoccupied—always allows a new car to come into the lot. We define these configurations as open nodes (See the green colored nodes in Fig. 4a). If any of the I/O cells are occupied, a new car cannot enter the parking lot which are called non-open nodes. There are five non-open nodes (grey nodes) in Fig. 4a: (11, 21), (11, 12), (21, 22), (21, 31), and (31, 41). Though (31, 41) does not immediately block the parking lot, it allows only a second car on (11, 21) and after that no more cars can enter the parking lot.

When a second car arrives at the I/O point, it becomes the initial node of a 2-car graph. Therefore, open nodes of the 1-car graph and initial nodes of the 2-car graph are connected by an entering car maneuver. Similarly, open nodes of the 2-car graph and initial nodes of the 3-car graph are connected. We call these connectors bridging edges. The bridging edges of 1 car and 2 car graphs are highlighted blue in Fig. 4c. We generalize, open nodes of the  $k$  car graph and initial nodes of the  $(k + 1)$ -car graph are connected using bridging edges. With bridging edges entered, we have one graph representing the entire state space (see Fig. 5 for the  $4 \times 4$  lot state space).

This state space graph has 5,913 vertices and 14,635 edges. There are 72 connected components, set of vertices reachable from one another, in this graph. Vertices from one connected component do not communicate with vertices from other connected components.

However, all vertices inside a connected component can reach one another. To better visualize the graph, we have annotated a few nodes. For example, Fig. 5a is a parking lot with only one car, Fig. 5d is a parking lot with four cars, and so on. To determine how many cars can drive into a parking lot, we begin by looking for vertices from the 1-car graph. The 1-car graph starts when the first car comes into the parking lot which is the root node. If we explore the root node's connected component, we can reach vertices up to 7-car graph. Remaining connected components do not start from the root node. Hence, it is not possible to reach these

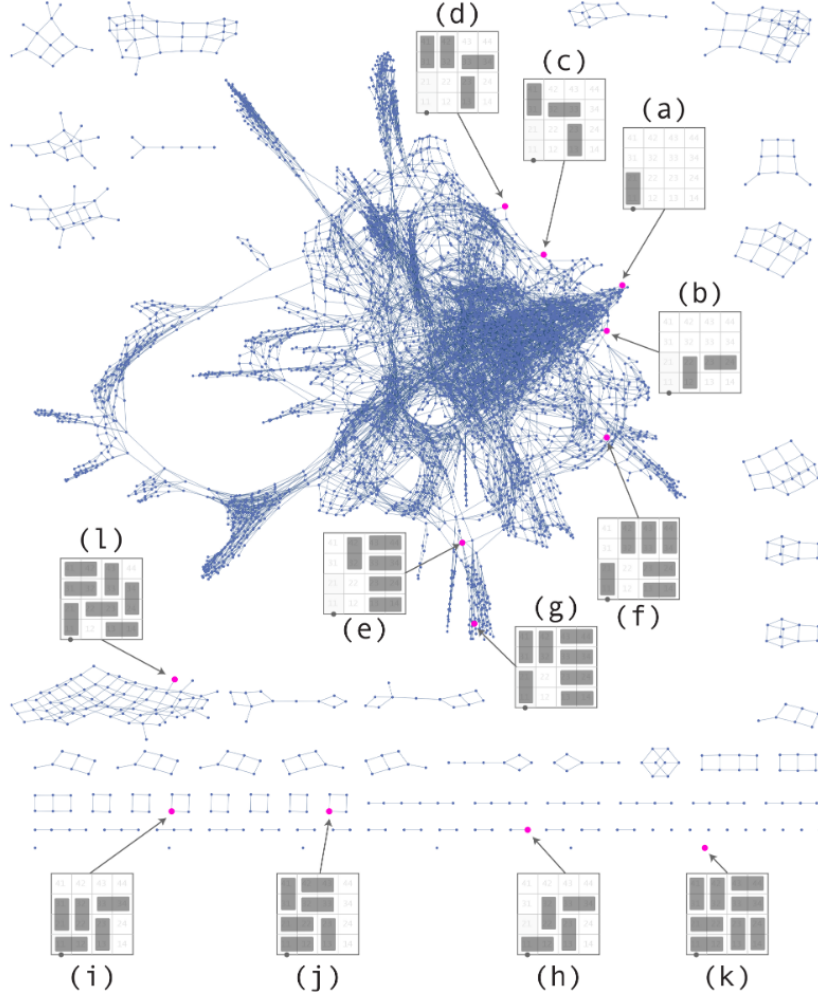


Fig. 5 State space graph of a  $4 \times 4$  parking lot. This graph has 5,913 vertices and 14,635 edges. Some of the vertices are annotated for the purpose of visualization.

configurations by driving the car. For example, vertices in Fig. 5i, 5j, 5k, 5l are not reachable from the root node. Therefore, we can park 7 cars in a  $4 \times 4$  parking lot at most. A parking lot with 7 cars is shown in Fig. 5g. In summary, though a  $4 \times 4$  parking lot has the capacity to hold 8 cars, we can only drive 7 cars into it. Steps to find the maximum number of cars which we can drive into an  $m \times n$  parking lot are as follows:

- Count the maximum number of cars that can be placed in a parking lot,  $k_{max} = \lfloor \frac{m \times n}{2} \rfloor$ .
- Generate the state space graphs for a parking lot with  $k$  cars, where  $k = 1, 2, \dots, k_{max}$ .
- For each of the graphs, identify initial nodes and open nodes.
- Bridge the open nodes of the  $k$  car graph with the initial nodes of the  $(k + 1)$  car graph. This yields a graph with a set of connected components.
- Find the connected component with the root node. This component is comprised of vertices from the 1 car graph to  $k_{le}$  car graphs, where  $k_{le} \leq k_{max}$ .
- Return the value of  $k_{le}$ , the maximum number of cars that can be driven into a parking lot.

We have learned that we can park a maximum of 7 cars in a  $4 \times 4$  parking lot. We can retrieve these 7 cars only in last-in, first-out sequence. However, always we do not want to leave a parking lot for the sole purpose

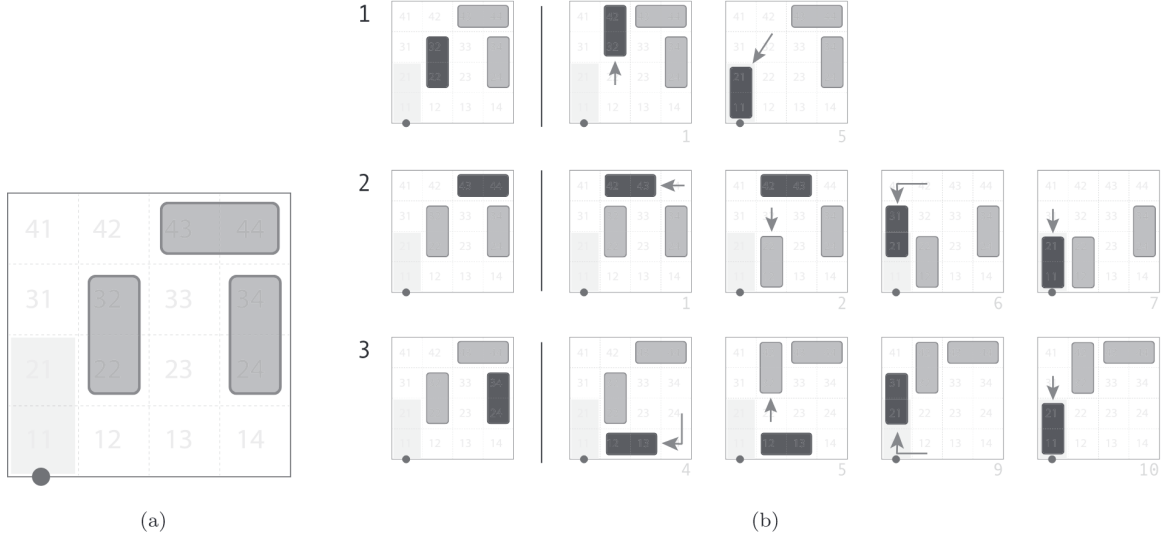


Fig. 6 Puzzle-based retrieval test. (a) A  $4 \times 4$  parking lot with 3 cars. (b) We perform Puzzle-based retrieval test on the layout on the left, and find the all the three cars can be retrieved by relocating other cars inside the parking lot.

of allowing another car to exit. Therefore, we want to know: how many cars can be parked in a parking lot, so that any car can depart while other cars remain? To answer this question, we select a parking lot with  $k$  cars, where  $k = 1, 2, \dots, 7$ , and test whether is it possible for any car to depart. As this test resembles the Rush Hour puzzle, we call this as puzzle-based retrieval test. For example, in Fig. 6 we perform a puzzle-based retrieval test on a parking lot with 3 cars. We observe that in all the three cases the target car reaches the I/O point and remaining cars do not need to leave the parking lot, but relocate to allow the target car to leave. Then, we perform the puzzle-based retrieval test on the entire state space to find the number of cars that can be parked in a  $4 \times 4$  parking lot. However, to minimize our search effort, we begin this test from parking lot configurations with 7 cars. We find that it is not possible to park 7 cars in this way. Then we explore configurations with 6 cars and the test fails again. Finally, for configurations with 5 cars, we find all the cars can leave. We do not need to continue this test for configurations with less than 5 cars, as surely there will be some configurations that will pass this test. Therefore, we find that a maximum of 5 cars can be parked in a  $4 \times 4$  parking lot, so that any car can depart while other cars remain.

In a traditional setting, we want to park such a way that any car can leave, without relocating other cars. Therefore, we want to know: how many cars can be parked in a parking lot, so that no car is blocked? To answer this question, we select a parking lot with  $k$  cars, where  $k = 1, 2, \dots, 5$ , and test whether it is possible for any car to depart the parking lot, without relocating other cars. We call this test as unblocked retrieval test. Then performing a brute-force operation, we find that a maximum 4 cars can be parked in a  $4 \times 4$  parking lot, so that no car blocks each other.

Observing the state space graph, we discover there exists three types of parking lot designs. The first type is a parking lot with maximum number of cars, cars completely block each other, and cars can only leave in last-in, first-out sequence. We call this design as limited egress (Fig. 7a). Such parking is commonly found in different events like, a concert or a ball game. Then the second type, where cars also block each other, but any car can leave by moving other cars, like the Rush Hour puzzle. We call this design as complete egress (Fig. 7b). And finally, the third one is the traditional parking lot (Fig. 7c), where no car blocks each other. We discovered these designs as we have initiated our research question without presupposing any particular

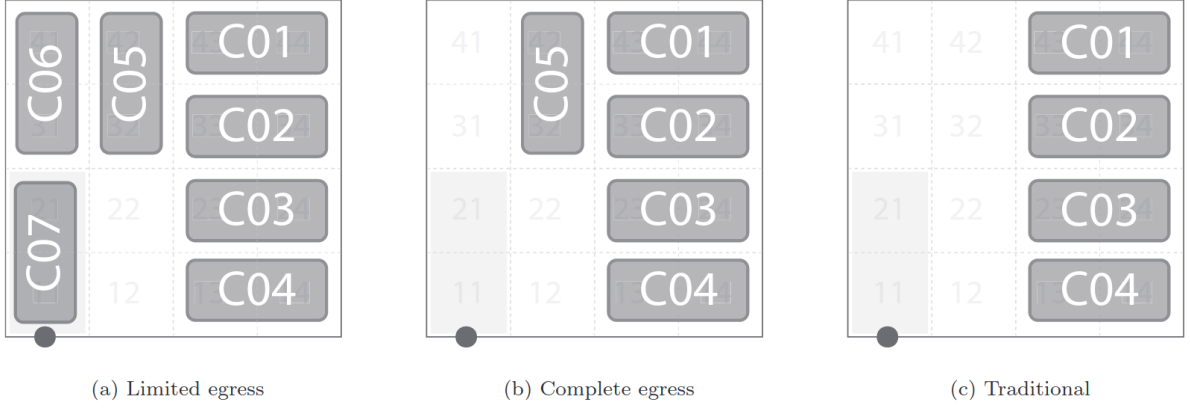


Fig. 7 Three parking lot designs.

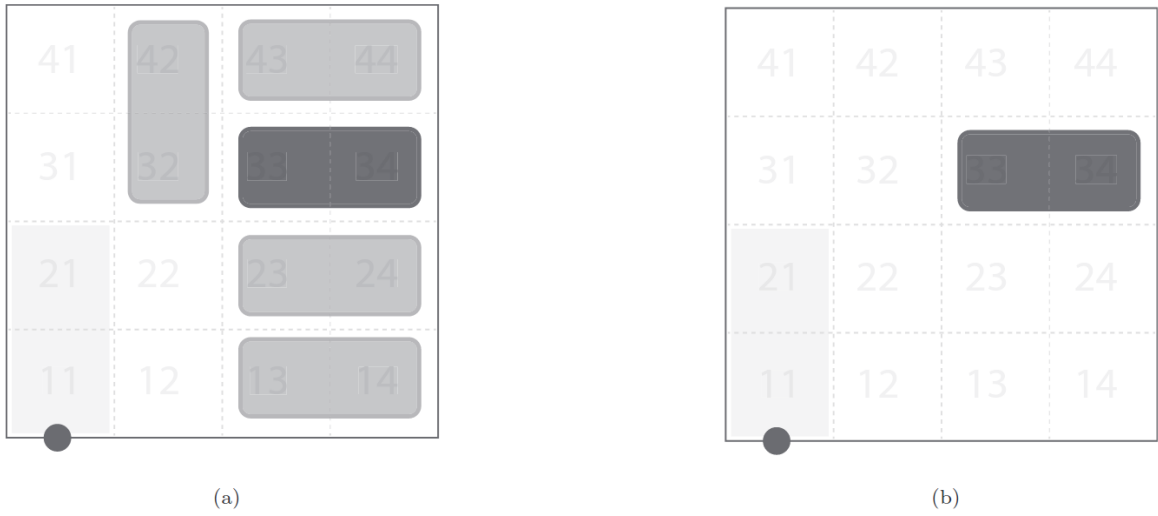


Fig. 8 Single-car heuristic for  $A^*$  search. To retrieve the dark colored car from the parking lot on the left, the parking lot on the right is used as heuristic by placing only one car on the same location as the dark colored car. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

layout or structure.

## 2.2 How long does it take to retrieve a car?

The retrieval time of each vehicle is one of the performance measures of these designs. We use the same state space graph to compute retrieval. We apply an  $A^*$  search algorithm on the state space graph to compute optimal retrieval of each of the cars in the parking lot.

$A^*$  is an informed search algorithm which explores a graph by expanding the most promising node chosen according to the smallest path cost. The path cost is evaluated as  $f(s) = g(s) + h(s)$ , where  $g(s)$  is the cost from the initial state to current state  $s$ , and  $h(s)$  is the estimated cost of the cheapest path from current state  $s$  to the goal state. Hart et al. also showed that  $A^*$  algorithm guarantees to produce optimal results if the heuristic function  $h(s)$  is admissible and consistent. An admissible heuristic,  $h(s)$ , is one that constructed in such a way that it generates lower bound to the real cost,  $h(s^*)$ , of reaching the goal states:  $h(s) \leq h(s^*)$ . A consistent heuristic is one in which for every state  $s$  and each successor state  $s'$  of  $s$ , the estimated cost,  $h(s)$ , of reaching the goal from  $s$  is no greater than the step cost,  $c(s, s')$ , of getting to  $s'$  plus the estimated cost,  $h(s')$ , of reaching the goal from  $s'$ , which can be expressed as  $h(s) \leq c(s, s') + h(s')$ . A consistent heuristic

is necessarily admissible while the reverse case need not be true. We have covered brief literature about  $A^*$  search, for more information readers are referred to Hart et al., Pearl, Russell and Norvig.

In our implementation of  $A^*$  algorithm, inputs are: a start node, a target car, and a heuristic function. The start node is the initial state which is represented by the size of a parking lot, number of cars parked and their locations. The target car is the car which we want to retrieve. As a heuristic function, we use the retrieval time of a single car. For example, in Fig. 8, to estimate the cost to retrieve the dark car from the parking lot on the left, we place only one car in the parking lot on the same location as the target car and compute retrieval time. The retrieval cost for the dark car is 27 (Fig. 8a) and heuristic cost is 5 (Fig. 8b). As it is a relaxed version of our problem, it will always be consistent and we can get the optimal result. We call this heuristic function a single-car heuristic.

Our retrieval algorithm explores the state space by using two lists: i) frontier and ii) explored. The frontier list is a priority queue that ranks all the nodes to be visited according to  $f(s)$  value and the explored list stores all the visited nodes. In the beginning, the frontier list contains only the initial node and the search stops when a goal node is chosen to visit. A goal node is a configuration where the target car is on the I/O point. We also create a third list called the parent-child which tracks relationships between the nodes—whenever our search reaches the goal node, this list is used in constructing the path from the initial node to the goal node. Applying  $A^*$  search, we have computed optimal number of moves required to retrieve vehicles from the limited egress, complete egress, and traditional parking lot in Fig. 9. Please note vehicle retrieval in a limited egress parking lot follows last-in, first-out sequence, we retrieved the 7<sup>th</sup> car first then the 6<sup>th</sup> and so on (see Fig. 7a), then computed cumulative retrieval time as shown in Fig. 9a.

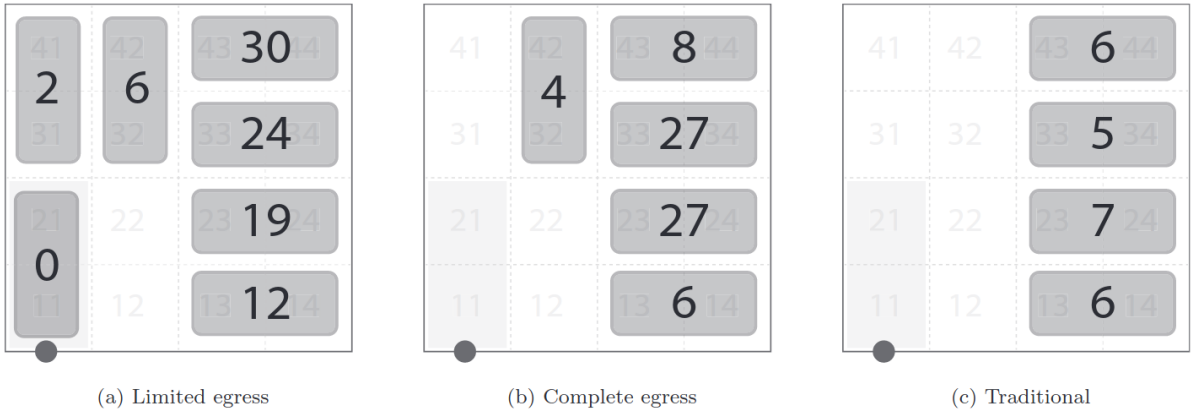


Fig. 9 Vehicle retrieval time.

Here, we have computed retrieval time assuming that a car takes one unit of time to travel one cell. However, reader may be interested to learn about retrieval time in a real parking lot. In this regard, we show an example here. In general, a parking spot dimension is  $9' \times 18'$ . As we have considered a parking spot  $1 \times 2$ , we can consider dimension of one cell as  $9' \times 9'$ . According to Schwesinger et al., speed limit for the autonomous vehicles in a parking lot is  $10\text{km/h} (\approx 9\text{ft/s})$ . Hence, travel time for one cell is 1 s. In the complete egress parking lot (Fig. 9b), to retrieve car 02, all the cars travel 27 cells. Therefore, retrieval time of the car 02 is 27 s.



## 译文 2

作者: Parag J. Siddique, Kevin R. Gue, John S. Usher

出处: Transportation Research Part C: Emerging Technologies, Volume 127, June 2021, 103112

### 基于谜题的停车

**摘要:** 在普通停车场中,大部分空间并非用于停车,而是用于车辆在停车位之间行驶的车道。车道不能被阻塞,一个简单的原因是:被阻塞的车可能需要在挡住它的车之前离开。自动停车和自动驾驶车辆的智能通信能力为克服这一限制提供了机会,因此可以实现更高的汽车存储容量。我们展示了如何通过集中控制器将干扰的车辆移开,从而最大化停车场中的汽车数量。我们提供了单入口小型停车场的最优结果,并为更大停车场提供了启发式方法。停车容量的提升可达 80%。

#### 1. 基于谜题的停车场

自主乘用车辆的前景为改善城市交通提供了许多有趣的机会。在停车方面,远程控制汽车的潜力允许汽车停放得更近,因为不需要打开车门让乘客下车。停车密度的提升潜力估计为 20%。另一个想法是,自动驾驶车辆不需要停在繁忙的城市区域。相反,可以在廉价的郊区创建远程停车区域。另一个被广泛接受的观点是,自动驾驶车辆将消除私人车辆所有权。自动出租车将带来共享出行,从而大幅减少停车需求。

在这一背景下,博世与梅赛德斯和福特合作,设计了一座为自动驾驶车辆设计的自动泊车场。到达停车场时,乘客在一个送车点离开车辆。停车场配备了智能传感系统,引导车辆找到停车位。每当乘客需要车时,可以通过智能手机应用程序检索车辆,车辆将到达取车点。值得注意的是,不需要人工控制汽车,也无需建设额外的市政基础设施。

在这样的自动化停车场中,通过消除行车道,可以实现更高的停车密度。在普通停车场中,行车道不能被封锁的一个简单原因是:被封锁的车可能需要在挡住它的车之前离开。然而,自动泊车场为克服这个问题,实现更高停车密度提供了机会。因此,我们已经拥有实现更高密度所需的技术,一切需要高效的设施设计和规划。

一些作者设想了一些停车方案,以提高停放汽车的密度,或等效地提高停车场的容量。Ferreira 等人首次提出了自主车辆的高密度停车方案。在他们的设计中,汽车通过在平行车道上相互阻挡来停放,类似于材料处理系统中使用的深车道存储系统(图1a)。他们假设有一个自主停车控制器,负责控制停车场内的车辆移动。这一组研究人员已将



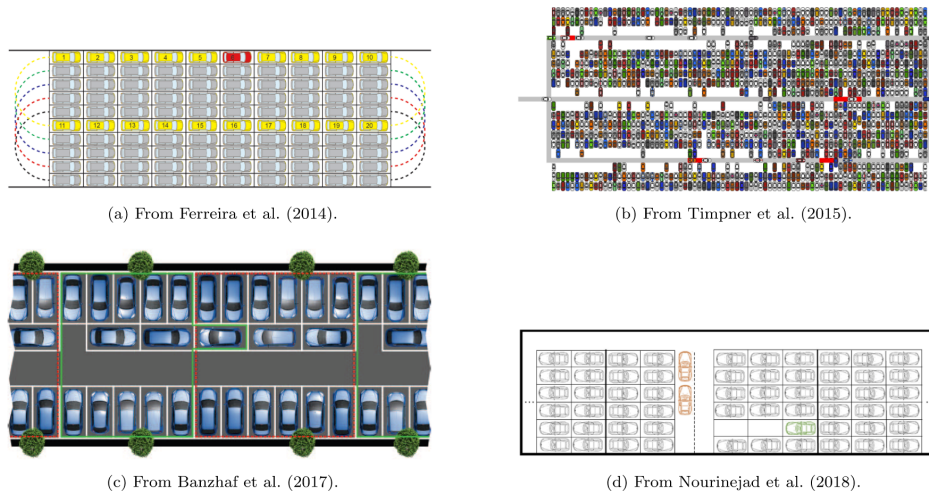


图 1 其他作者提出的高密度停车设计。(a) Ferreira 等人 (2014) 提出了一种通过在平行车道上相互阻挡的设计来停放汽车。他们在设施的入口和出口都分配了缓冲区域，以便车辆可以进行必要的移动进行重新定位。(b) Timpner 等人在被阻挡的车道之间放置了垂直过道，以减少干扰车辆的数量。阻挡的车辆重新定位到过道上，为即将离开的车辆清出道路。(c) Banzhaf 等人通过添加一个车辆阻挡的过道来修改现有的停车场。(d) 在 Nourinejad 等人的设计中，过道的宽度随着车道的深度而变化，以帮助容纳重新定位的车辆。

这项工作扩展到考虑各种操作参数，如每辆车的停车空间、操纵次数、行驶距离和检索时间。他们的设计将停车容量提高了 50%。

为了减少干扰车辆的数量，Timpner 等人修改了 Ferreira 等人的设计，减小了车道深度，并允许车辆从车道的任一端离开。作者估计密度提高了 33%。Banzhaf 等人建议通过在现有过道内允许垂直的阻挡车道来修改现有布局。他们报告密度增加了 25%。Nourinejad 等人描述了一个类似的系统，其中过道的宽度随着车道的深度变化，以帮助容纳重新定位的车辆。作者制定了一个混合整数非线性规划来最小化预期的车辆重新定位次数。然而，所有这些研究人员都计划针对更适合郊区地区的大型停车场。

从文献中我们了解到，自主车辆不会在停车场闲置。在放下乘客后，它将用于运送另一名乘客。然而，由于供需之间存在差距，当没有行程时会有一些空闲时间，因此车辆必须停在某个地方。如果这些车辆去郊区停车，它们将无法响应客户的呼叫。因此，自主车辆不会选择去城外停车，而是会在城市中心周围巡游以打发时间，从而导致城市拥堵。因此，我们无法完全消除城市停车。相反，我们可以寻找使停车更容易访问和更便宜的方法。为此，我们可以为城市区域提供小型而密集的停车场。这些停车场可以与零售业中的微型履行中心相类比。微型履行中心是靠近客户的小型仓库，专为按需电子商务履行而设计。每当自主车辆处于空闲状态时，它可以停在这些小型停车场。由于这些停车场将位于城市区域，因此每当车辆接收到行程请求时，它可以迅速服务请求。作为增值服务：在这些停车场中还可以引入车辆充电、清洁或类似的服务。

在本文中，我们研究了自主车辆停车场的高密度布局设计。我们的工作在两个重要

方面与现有文献有所不同：

1. 我们为小型停车场开发最优设计，这是人们可能在城市环境中找到的。
2. 所有现有的研究都是基于一个布局，然后研究在该布局下的最优参数和性能。我们提出并回答了更基本的问题，即在不假设任何特定布局或结构的情况下，停车场可以停放多少辆车。正如我们将展示的，这种放宽导致了具有最大密度的设计，而不采用传统的“排和过道”停车结构。



图 2 Rush Hour 难题。游戏的目标是通过移动其他车辆让红车直接移动到对面的出口。

我们处理的问题类似于 Rush Hour 难题（图2），这在理论计算机科学领域有相当多的文献。Flake 和 Baum 表明，具有  $n \times n$  网格的 Rush Hour 的广义版本是 PSPACE-complete。Tromp 和 Cilibrasi 以及 Hearn 和 Demaine 表明，仅包含汽车（没有  $3 \times 1$  的卡车）的 Rush Hour 也是 PSPACE-complete。我们的问题在很多方面都与 Rush Hour 不同：游戏只允许车辆前后移动，而真实汽车可以转弯；而 Rush Hour 的目标是移走一个特定的车辆，而真实的停车场必须允许任何请求的车辆到达或离开。在物料处理文献中，也已经讨论了没有定义过道的高密度存储，其中使用可以沿着四个基本方向移动的输送模块存储和检索单元负载或容器。然而，在我们的停车场问题中，汽车独自移动，不需要任何输送模块的帮助，而且汽车可以朝不同的方向行驶。

## 2. 建模基于谜题的停车场

我们将停车场视为一个矩形网格。汽车可以水平或垂直停放。我们假设汽车在网格中占据两个单元格，这意味着汽车的大小为  $1 \times 2$ 。停车场的左下角有一个入口/出口点。

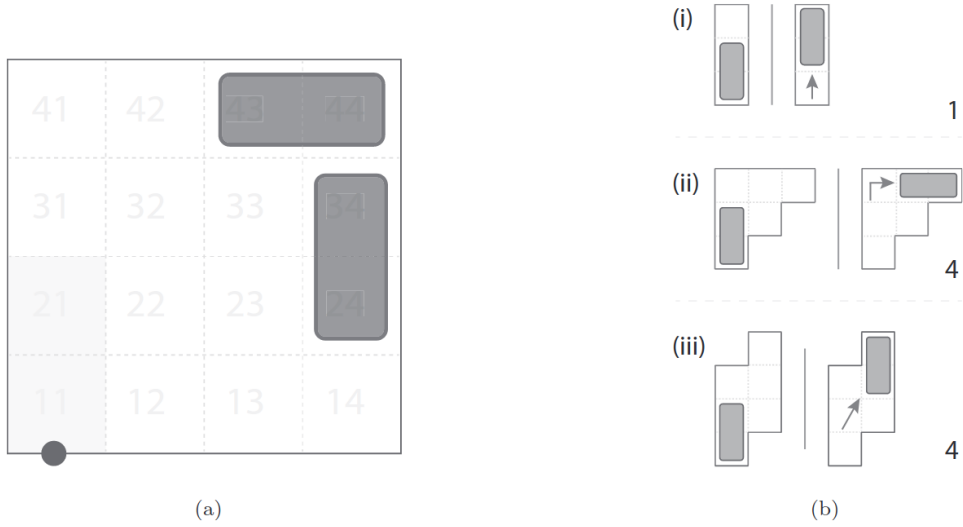


图 3 (a) 一个  $4 \times 4$  停车场的示例，其中有两辆车，一辆水平停放在单元格 (43, 44)，另一辆垂直停放在单元格 (23, 34)，而单元格 (11, 21) 是输入/输出点。(b) 停车场内的三个有效移动。每个移动根据所行单元格的数目分配一个权重。

我们称这个点为输入-输出 (I/O) 点。在图3a 中，两辆车停在一个  $4 \times 4$  停车场中，一辆停在单元格 (43, 44)，另一辆停在单元格 (24, 34)。单元格 (11, 21) 表示 I/O 点。

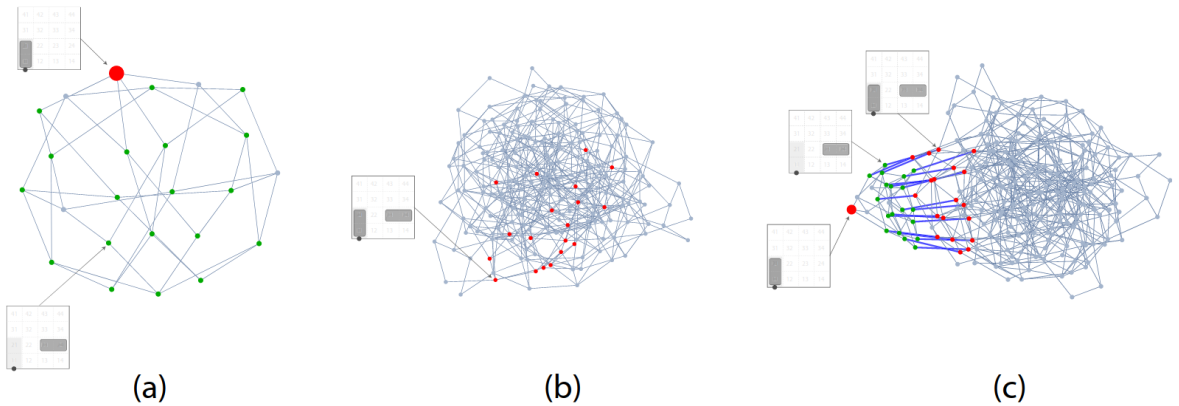


图 4 生成  $4 \times 4$  停车场状态空间图的过程。(a) 单车图。大红色节点是根节点—这个图中的第一个节点。绿色节点是开放节点。开放节点总是允许新车进入停车场。根节点和一个开放节点都已注释。(b)  $4 \times 4$  网格中的 2 车图。红色节点是初始节点，包括 I/O 单元格。一个初始节点已注释。(c) 连接单车图和 2 车图。单车图中的开放节点连接到 2 车图中的初始节点，用蓝色突出显示。这些连接器称为桥接边缘。先前注释的节点的位置也显示出来。

车辆在停车场内可以进行三种类型的移动：直行、直角和平行，这些操作可以前进或后退完成（图3b）。假设车辆花费一个单位的时间在一个单元格的行驶，可以为每个移动分配一个权重以量化行驶时间。例如，车辆进行直行时移动一个单元格，因此将此移动分配一个权重为一。类似地，车辆进行直角和平行移动时移动四个单元格，因此将这些移动分配一个权重为四。我们假设停车场中有一个中央控制器代理，可以与车辆通信以执行所有移动。

读者可能会注意到这里我们并没有指定单元格的尺寸。单元格的大小是根据车辆的尺寸和运动学来决定的。相反，我们希望将单元格的设计留给专业人士。

## 2.1 停车场能容纳多少辆车？

我们使用状态空间图  $G = (V, E)$  对停车场中的车辆进行建模，其中  $V$  是表示停车场配置的顶点集合， $E$  是表示配置之间可行转换的边缘集合。在本文中，顶点也可以用术语“节点”来表示。配置被定义为停车场的尺寸，停放的车辆数量以及它们的位置。术语“状态”也被用作配置的替代术语。我们想知道：停车场能停放多少辆车？为了开始解决这个问题，我们解决一个小例子，即  $4 \times 4$  停车场。

在一个  $4 \times 4$  停车场中，有十六个单元格，每辆车占用两个单元格。因此，我们最多可以放置八辆车。我们首先为  $4 \times 4$  网格生成状态空间图。在不考虑可操纵性的情况下，一个单独的 vehicle 可以放置在 24 个不同的位置，每个位置都代表图中的一个潜在节点。然后，我们对所有潜在节点执行两两搜索，确定它们是否可以通过可行的操纵连接。我们将结果称为单车图（图4a）。接下来，我们以每种可能的方式放置两辆车，以创建 2 车图。然后我们消除了具有重叠车辆的配置。例如，配置  $[(11, 21), (21, 22)]$  是不可行的，因为单元格 21 被两辆车使用，这在物理上是不可能的。我们还消除了占据整行或整列的配置，因为它们是不可行的（除了  $[(11, 21), (31, 41)]$ ）。最后，我们使用它们各自的移动连接节点，构建了完整的 2 车图（图4c）。同样，我们可以继续构建多达八辆车的图。

我们发现一个有趣的特征来连接这八个图。在一个空的停车场中，当第一辆车到达时，它最初停放在 I/O 单元格上。这是单车图的起始节点。这辆车可以移动到停车场内的不同位置。一旦车辆离开 I/O 单元格，停车场就可以接收第二辆车。当第二辆车到达时，它将自己停放在 I/O 单元格上。这是 2 车图的起始点。之后，车辆可以重新定位到停车场内的其他位置。类似地，当这两辆车都停在 I/O 单元格之外的任何位置时，停车场就可以接收第三辆车，我们可以一直进行此过程直到 8 辆车。因此，我们观察到每个  $k$  车图（ $k = 1, 2, \dots, 8$ ）都从 I/O 单元格开始。因此，我们将包括占用的 I/O 单元格的停车场配置定义为初始节点。2 车图的初始节点在图4b 中以红色突出显示。请注意，每个  $k$  车图都有多个初始节点，除了 1 车图。为了将此节点与其他初始节点区分开，我们将其定义为根节点（见图4a 中的大红色节点）。我们还观察到为了容纳新车辆进入停车场，我们必须确保没有车辆停在 I/O 单元格上。这样的配置——I/O 单元格未被占用——始终允许新车辆进入停车场。我们将这些配置定义为开放节点（见图4a 中的绿色节点）。如果 I/O 单元格中的任何一个被占用，新车辆将无法进入停车场，这些节点称为非开放节点。在图4a 中有五个非开放节点（灰色节点）： $(11, 21)$ 、 $(11, 12)$ 、 $(21, 22)$ 、 $(21, 31)$  和  $(31, 41)$ 。尽管  $(31, 41)$  不会立即阻塞停车场，但它只允许第二辆车停在  $(11, 21)$  上，之

后不能再有车辆进入停车场。

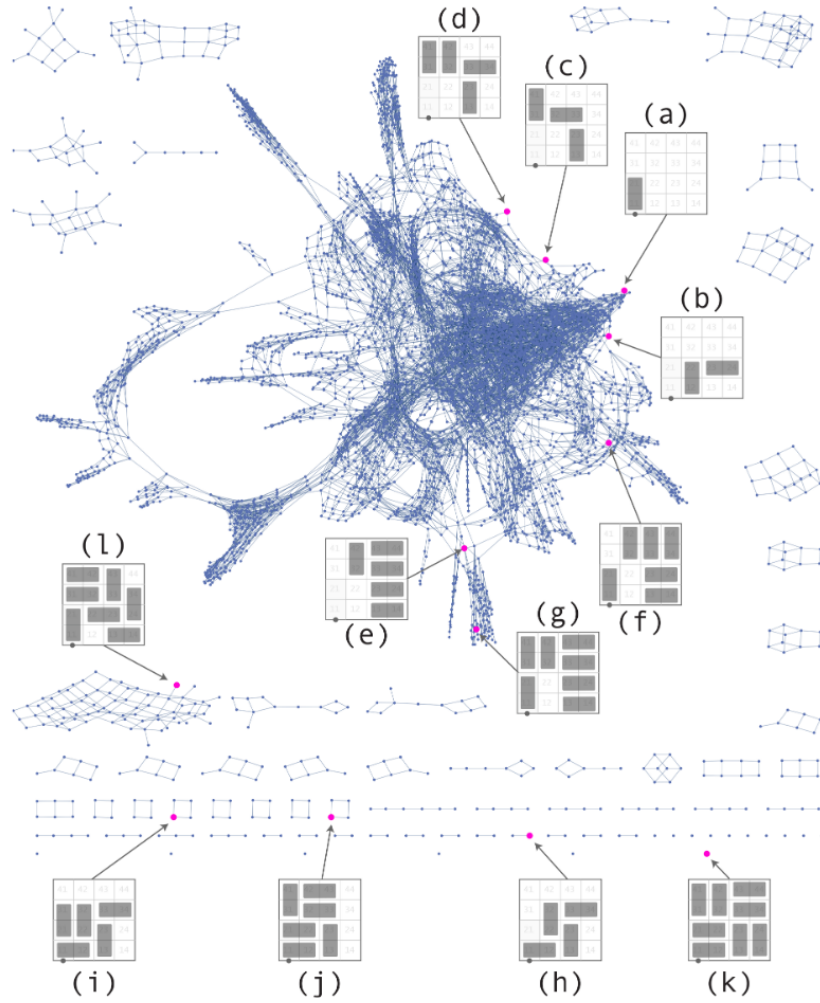


图5  $4 \times 4$  停车场的状态空间图。此图具有 5,913 个顶点和 14,635 条边。为了可视化目的，已注释其中一些顶点。

当第二辆车到达 I/O 点时，它成为 2 车图的初始节点。因此，1 车图的开放节点和 2 车图的初始节点通过一种进入车辆的操纵连接在一起。同样，2 车图的开放节点和 3 车图的初始节点也是如此。我们称这些连接器为桥接边缘。1 车和 2 车图的桥接边缘在图4c 中用蓝色突出显示。我们一般化， $k$  车图的开放节点和  $(k + 1)$ -车图的初始节点使用桥接边缘连接。有了桥接边缘，我们就有了一个代表整个状态空间的图（请参见图5，显示了  $4 \times 4$  停车场的状态空间）。

该状态空间图具有 5,913 个顶点和 14,635 条边。在此图中，有 72 个连接组件，即彼此可达的顶点集。来自一个连接组件的顶点与来自其他连接组件的顶点之间不进行通信。

然而，连接组件内的所有顶点都可以相互到达。为了更好地可视化图，我们已注释了一些节点。例如，图5a 是一个只有一辆车的停车场，图5d 是一个有四辆车的停车场，依此类推。为了确定有多少辆车可以驶入停车场，我们从寻找 1 车图的顶点开始。1 车



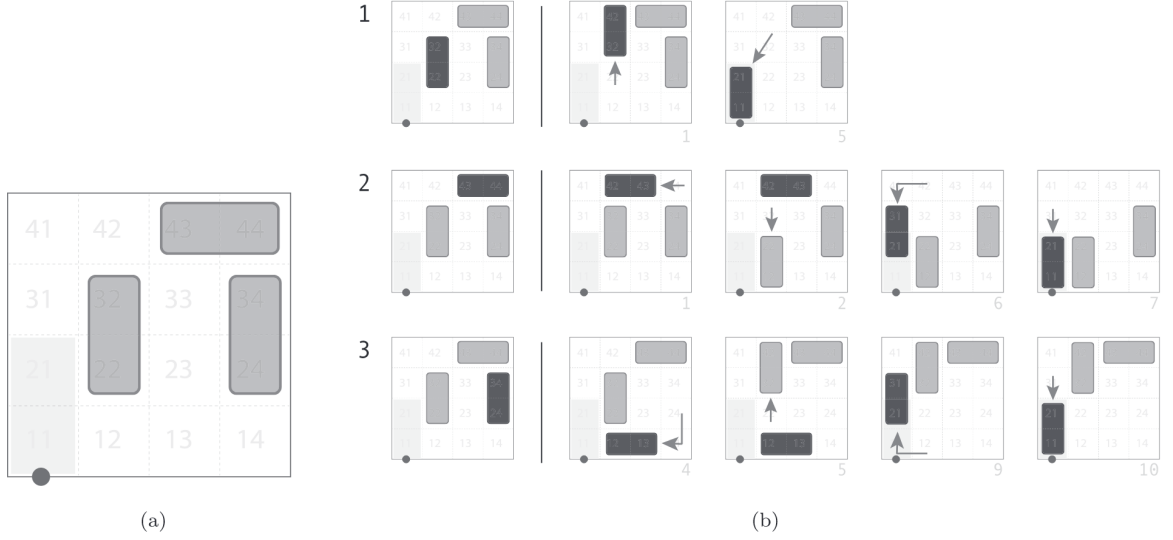


图 6 基于谜题的检索测试。(a) 一个带有 3 辆车的  $4 \times 4$  停车场。(b) 我们在左侧的布局上执行基于谜题的检索测试，并发现通过重新定位停车场内的其他车辆，可以检索出所有三辆车。

图始于第一辆车进入停车场的时刻，即根节点。如果我们探索根节点的连接组件，我们可以达到高达 7 车图的顶点。其余的连接组件不是从根节点开始的。因此，不可能通过驾驶车辆达到这些配置。例如，图5i、图5j、图5k 和图5l 中的顶点不能从根节点到达。因此，在  $4 \times 4$  停车场中最多可以停放 7 辆车。图5g 显示了一个有 7 辆车的停车场。总之，尽管  $4 \times 4$  停车场有容纳 8 辆车的能力，但我们只能驾驶 7 辆车进入。查找我们可以驾驶进入  $m \times n$  停车场的最大车辆数的步骤如下：

- 计算可以放置在停车场中的最大车辆数， $k_{max} = \lfloor \frac{m \times n}{2} \rfloor$ 。
- 为具有  $k$  辆车的停车场生成状态空间图，其中  $k = 1, 2, \dots, k_{max}$ 。
- 对于每个图，识别初始节点和开放节点。
- 将  $k$  车图的开放节点与  $(k + 1)$  车图的初始节点连接起来。这将产生一个具有一组连接组件的图。
- 找到带有根节点的连接组件。该组件由 1 车图到  $k_{le}$  车图的顶点组成，其中  $k_{le} \leq k_{max}$ 。
- 返回  $k_{le}$  的值，即可以驾驶进入停车场的最大车辆数。

我们已经了解到，在  $4 \times 4$  停车场中，我们最多可以停放 7 辆车。我们只能以后进先出的顺序检索这 7 辆车。然而，我们并不总是希望为了让另一辆车离开而离开停车场。因此，我们想知道：在停车场中最多可以停放多少辆车，以便其中的任何一辆车在其他车辆保留的情况下离开？为了回答这个问题，我们选择一个有  $k$  辆车的停车场，其中

$k = 1, 2, \dots, 7$ , 并测试是否有可能让任何一辆车离开。由于这个测试类似于 Rush Hour 拼图, 我们称之为基于谜题的检索测试。例如, 在图6中, 我们对一个有 3 辆车的停车场进行了基于谜题的检索测试。我们观察到在所有三种情况下, 目标车辆达到了 I/O 点, 而其余车辆不需要离开停车场, 而是重新定位以使目标车辆离开。然后, 我们对整个状态空间进行基于谜题的检索测试, 以找到可以停放在  $4 \times 4$  停车场中的车辆数。然而, 为了最小化我们的搜索工作量, 我们从具有 7 辆车的停车场配置开始这个测试。我们发现这样停放 7 辆车是不可能的。然后我们探索具有 6 辆车的配置, 测试再次失败。最后, 对于具有 5 辆车的配置, 我们发现所有车辆都可以离开。我们不需要对具有少于 5 辆车的配置继续进行这个测试, 因为肯定会有一些配置通过这个测试。因此, 我们发现在  $4 \times 4$  停车场中, 最多可以停放 5 辆车, 以便其中的任何一辆车在其他车辆保留的情况下离开。

在传统设置中, 我们希望停放的方式是任何一辆车都可以离开, 而不需要重新定位其他车辆。因此, 我们想知道: 在停车场中最多可以停放多少辆车, 以便没有车辆被阻挡? 为了回答这个问题, 我们选择一个有  $k$  辆车的停车场, 其中  $k = 1, 2, \dots, 5$ , 并测试是否有可能让任何一辆车离开停车场, 而不需要重新定位其他车辆。我们将这个测试称为未阻挡检索测试。然后, 通过进行蛮力操作, 我们发现在  $4 \times 4$  停车场中, 最多可以停放 4 辆车, 以便没有车辆相互阻挡。

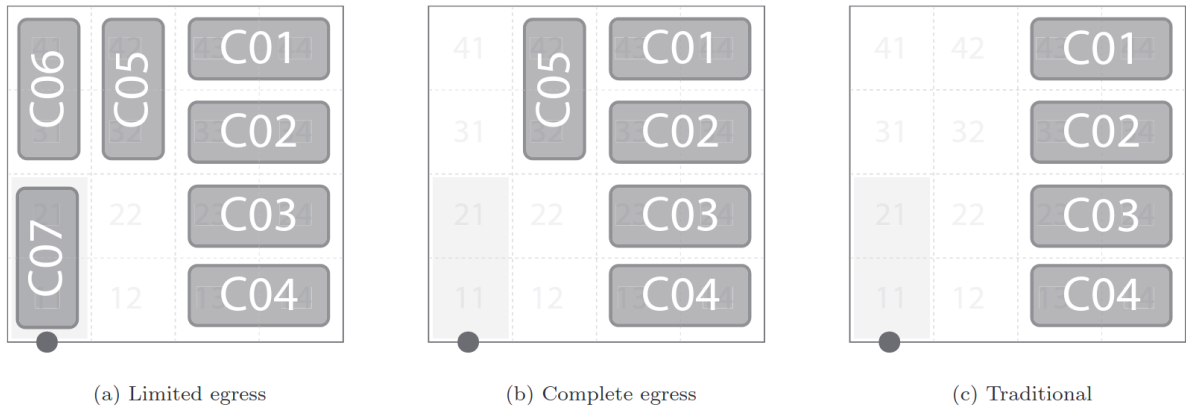


图 7 三种停车场设计。

观察状态空间图, 我们发现存在三种停车场设计。第一种类型是停车场, 最多停放车辆, 车辆完全相互阻挡, 车辆只能按照后进先出的顺序离开。我们将这种设计称为有限出口 (图7a)。这种停车场通常在不同的活动中发现, 如音乐会或球赛。然后是第二种类型, 车辆也相互阻挡, 但任何车辆都可以通过移动其他车辆离开, 就像 Rush Hour 拼图一样。我们将这种设计称为完全出口 (图7b)。最后, 第三种是传统停车场 (图7c), 在这种停车场中, 没有车辆相互阻挡。我们发现了这些设计, 因为我们在提出研究问题

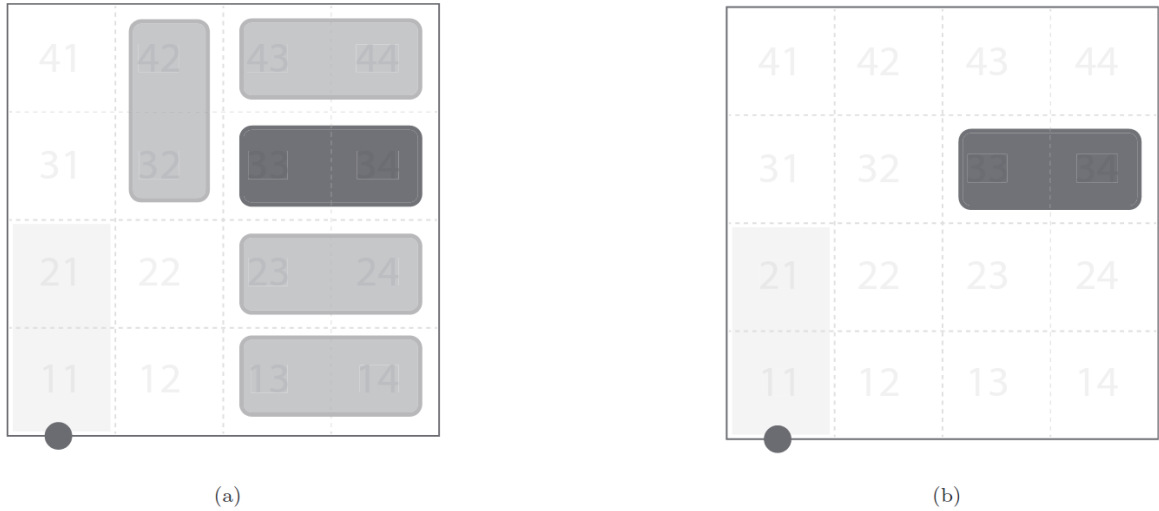


图8  $A^*$  搜索的单车启发式。为了从左侧停车场中检索深色车辆，右侧停车场被用作启发式，只在与深色车辆相同的位置放置一辆车。（有关本图例中颜色的解释，请参阅本文的网络版本。）

时没有预设任何特定的布局或结构。

## 2.2 检索车辆需要多长时间？

每辆车的检索时间是这些设计的性能指标之一。我们使用相同的状态空间图来计算检索时间。我们在状态空间图上应用  $A^*$  搜索算法来计算停车场中每辆车的最佳检索。

$A^*$  是一种启发式搜索算法，通过扩展根据最小路径成本选择的最有前途的节点来探索图。路径成本评估为  $f(s) = g(s) + h(s)$ ，其中  $g(s)$  是从初始状态到当前状态  $s$  的成本， $h(s)$  是从当前状态  $s$  到目标状态的最便宜路径的估计成本。Hart 等人还表明，如果启发函数  $h(s)$  是可接受的和一致的，则  $A^*$  算法保证产生最优结果。一个可接受的启发式  $h(s)$  是这样构建的，以便它生成到达目标状态的实际成本  $h(s^*)$  的下界： $h(s) \leq h(s^*)$ 。一致的启发式是这样的，对于每个状态  $s$  和  $s$  的每个后继状态  $s'$ ，从  $s$  到目标的估计成本  $h(s)$  不大于到  $s'$  的步骤成本  $c(s, s')$  加上从  $s'$  到目标的估计成本  $h(s')$ ，可以表达为  $h(s) \leq c(s, s') + h(s')$ 。一致的启发式必然是可接受的，而反之则未必成立。我们已经简要介绍了  $A^*$  搜索的文献，更多信息请参阅 Hart 等人，Pearl, Russell 和 Norvig。

在我们的  $A^*$  算法实现中，输入包括：起始节点、目标车辆和启发式函数。起始节点是初始状态，由停车场的大小、停放的车辆数量和它们的位置表示。目标车辆是我们检索的车辆。作为启发式函数，我们使用单车检索时间。例如，在图8中，为了估算从左侧停车场检索深色车辆的成本，我们在右侧停车场的与目标车辆相同的位置放置一辆车，并计算检索时间。深色车辆的检索成本为 27（图8a），启发成本为 5（图8b）。由于这是我们问题的简化版本，它始终是一致的，因此我们可以得到最优结果。我们将这个启发式函数称为单车启发式。



我们的检索算法通过使用两个列表来探索状态空间：i) **frontier**（前沿）和 ii) **explored**（已探索）。前沿列表是一个按照  $f(s)$  值对所有待访问节点进行排名的优先级队列，而已探索列表则存储所有已访问的节点。在开始时，前沿列表只包含初始节点，当选择访问目标节点时搜索停止。目标节点是目标车辆位于 I/O 点的配置。我们还创建了一个称为父-子的第三个列表，用于跟踪节点之间的关系——每当我们的搜索达到目标节点时，此列表用于构建从初始节点到目标节点的路径。应用  $A^*$  搜索，我们已经计算出从图9中的有限出口、完全出口和传统停车场检索车辆所需的最佳移动次数。请注意，在有限出口停车场中，车辆的检索遵循后进先出的顺序，我们首先检索第 7 辆车，然后是第 6 辆车，依此类推（见图7a），然后计算累积检索时间，如图9a 所示。

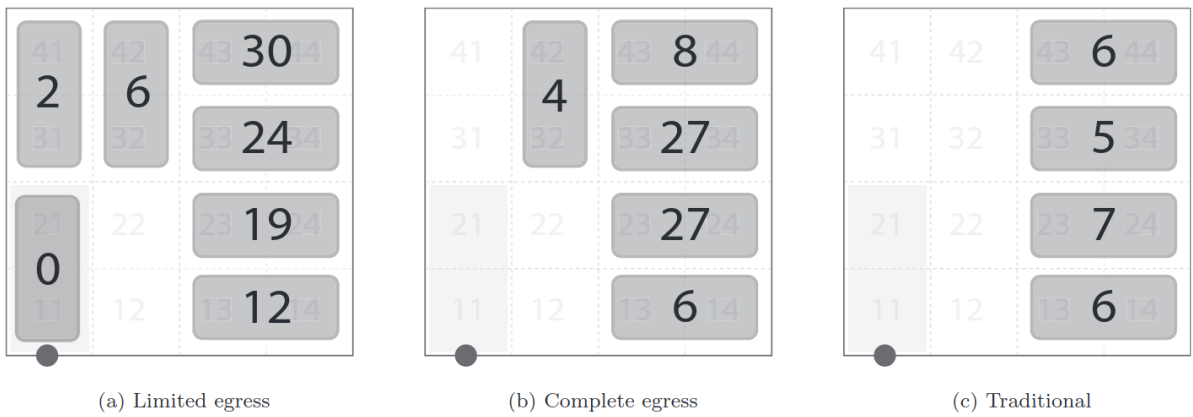


图 9 车辆检索时间。

在这里，我们计算检索时间，假设车辆在一个单元格内行驶需要一个时间单位。然而，读者可能对在真实停车场中的检索时间感兴趣。在这方面，我们在这里展示一个例子。一般而言，一个停车位的尺寸是  $9' \times 18'$ 。由于我们考虑的是一个  $1 \times 2$  的停车位，我们可以将一个单元格的尺寸视为  $9' \times 9'$ 。根据 Schwesinger 等人的说法，停车场中自动驾驶车辆的速度限制为  $10km/h (\approx 9ft/s)$ 。因此，一个单元格的行驶时间为 1 秒。在完全出口的停车场（图9b）中，为了取回车辆 02，所有车辆共行驶 27 个单元格。因此，车辆 02 的检索时间为 27 秒。