计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202440190
61(8): 1945−1956, 2024

Graph4Cache: 一种用于缓存预取的图神经网络模型

尚 晶 武智晖 肖智文 张逸飞
(中国移动信息技术中心 北京 100033)
(中移动信息技术有限公司 北京 100033)
(shangjing@chinamobile.com)

Graph4Cache: A Graph Neural Network Model for Cache Prefetching

Shang Jing, Wu Zhihui, Xiao Zhiwen, and Zhang Yifei
(*China Mobile Information Technology Center*, *Beijing* 100033)
(*China Mobile Information Technology Co.*, *Ltd.*, *Beijing* 100033)

Abstract Most computing systems utilize caching to reduce data access latency, speed up data processing and balance service load. The key to cache management is to determine the appropriate data to be loaded into or discarded from the cache, as well as the appropriate timing for cache replacement, which is critical to improving cache hit rate.The existing caching schemes face with two problems: In real-time and online caching scenarios, it is difficult to discern the heat information of user access to data while ignoring the complex high-order information among data-access-sequences. In this paper, we propose a GNN-based cache prefetching network named Graph4Cache. We model a single access sequence into a directed graph (ASGraph), where virtual nodes are used to aggregate the features of all nodes in graph and represent the whole sequence. Then a cross sequence undirected graph (CSGraph) is constructed from the virtual nodes of ASGraphs to learn cross-sequence features, which greatly complements the limited item transitions in a single sequence. By fusing the information of these two graphs , we learn the high-order correlations among sequences and get abundant user intents. Experimental results on multiple public data sets demonstrate the effectiveness of this method. Graph4Cache outperforms the existing cache prediction algorithms on *P*@20 and *MRR*@20.

Key words graph neural network; cache prefetching; access sequence graph; cross sequence graph; cache prediction

摘 要 大多数计算系统利用缓存来减少数据访问时间,加快数据处理并平衡服务负载.缓存管理的关键在于确定即将被加载到缓存中或从缓存中丢弃的合适数据,以及进行缓存置换的合适时机,这对于提高缓存命中率至关重要.现有的缓存方案面临2个问题:在实时的、在线的缓存场景下难以洞察用户访问数据的热度信息,以及忽略了数据访问序列之间复杂的高阶信息.提出了一个基于GNN的缓存预取网络Graph4Cache.通过将单个访问序列建模为有向图(ASGraph),并引入虚拟节点聚合图中所有节点的信息和表示整个序列.然后由ASGraph的虚拟节点构造一个跨序列无向图(CSGraph)来学习跨序列特征,这极大地丰富了单个序列中有限的数据项转换模式.通过融合这2种图结构的信息,学习到了序列之间的高阶关联信息,并获取了丰富的用户意图.在多个公共数据集上的实验结果证明了该方法的有效性.Graph4Cache在*P*@20和*MRR*@20上均优于现有的缓存预测算法.

关键词 图神经网络;缓存预取;访问序列图;跨序列图;缓存预测

收稿日期:2024-03-15;**修回日期**:2024-05-24

基金项目:国家重点研发计划项目(2023YFB4503100);国家自然科学基金项目(U23B2027);中国移动集团战略研发项目(R24113FN)
This work was supported by the National key Research and Development Program of China (2023YFB4503100), the National Natural Science Foundation of China (U23B2027), and the Strategic Research and Development Project of China Mobile (R24113FN).

通信作者:肖智文(xiaozhiwen@chinamobile.com)

中图法分类号 TP391

在存储系统有限的资源配置中,缓存是一种用于提升系统效率和性能的重要技术,它将频繁访问的热点数据和关键数据临时存储在快速可访问的位置,从而加快数据检索和处理速度[1]. 为了应对来自不同客户的实时变化的访问需求,缓存中的数据往往是不断更新的. 因此,如何从大量的数据访问请求中提取出一般规律,从而确定被更新的缓存数据项和更新的时机成为提高缓存命中率的关键.

传统的基于规则的缓存更新策略如FIFO、LRU、LFU等,仅依赖访问序列中数据的访问频率和最近访问时间等信息来选择丢弃的数据,并未主动对热点数据进行预取,也忽视了全局或长期的数据热度问题. 随着深度学习技术的演进和应用,基于深度学习的缓存预测方法逐渐开始受到关注. 这类方法通过分析全局数据访问热度信息,基于深度学习的缓存替换策略可以更准确地预测数据访问特征,从而提高缓存策略的有效性[2-3]. 然而这类预测模型大多依赖于充足的历史访问数据来训练和调整用户表示. 在实际缓存预测场景中,用户的访问模式可能是稀疏的,特别是对于新添加的或不常见的数据项,可能没有足够的历史访问记录来构建准确的表示. 另一方面,用户的兴趣往往是一个长期的、动态变化的过程,基于深度学习的缓存难以从大量访问数据中准确提取用户意图.

图神经网络(graph neural networks, GNNs)代表了深度学习领域的前沿技术. 得益于其对非欧几里得数据结构的强大处理能力,GNN在涉及单个或多个图结构的任务中表现出卓越的性能,近年来随着网络结构化数据的增多,基于GNNs的模型在网络分析领域得到了广泛应用[4-6]. 将数据访问序列建模为有向图结构后,可以使用GNN将图结构数据转换为低维、高密度的向量表示,同时学习图中节点之间的相连关系,进而捕捉序列中数据项之间复杂的转换模式,进行推理和预测. 大多基于GNN的顺序推荐方法只考虑同一个序列中相邻数据项之间的关系,这种做法忽略了较远的数据项的联系,因此无法处理较长的访问序列[7-9],同时也没有考虑序列之间高阶的相关信息[9-11]. 也有研究通过构建和学习全局图或异构图等结构来捕捉访问序列之间或用户与数据项之间的复杂关系,但这样的构图方式相对而言较为复杂且构图和计算成本较大[12-14].

在此项工作中,我们将基于GNN的推荐算法应用于缓存替换策略,提出了Graph4Cache. 考虑到当前基于GNNs的推荐方法主要在相邻数据项(item)之间传播信息,忽略了非相邻数据项的信息,而用户的访问兴趣却需要考虑长期访问行为. 因此,我们通过引入虚拟节点考虑网络中的非相邻项,解决了长序列的问题. 这种方法能够更有效地从用户的数据访问序列中提取用户偏好和数据项之间的转换模式,从而确定缓存预取内容. 同时,考虑现有的通过构建用户表示来进行预测的方法很容易受到历史访问数据不足的限制,我们不再区分不同用户的访问序列,而是为每一个单独的序列构建一个表示,这样就可以避免需要从同一个用户大量的历史访问序列中学习该用户的表示. 同时通过考虑每个序列和全局所有其他序列的关系来构建全局图,从而丰富了单个序列中有限的数据转换模式,并解决了单个用户访问模式稀疏的问题. 提出的方法构建了访问序列图和跨序列图,分别学习2个层面的嵌入表示:1)在序列层面,为单个序列构建访问序列图,并引入一个虚拟节点来聚合序列中所有数据项的表示,学习每个项的嵌入表示和整个序列的初步表示. 2)在全局层面,取访问序列图的虚拟节点作为该序列的嵌入表示,并构建跨序列图,其中每个节点代表一个序列,从而学习序列之间的高阶信息.

本文的主要贡献有4个方面:

1)将序列推荐的思想应用于缓存预取任务,并将数据访问序列建模为有向图,实现了通过图神经网络提高缓存命中率.

2)采用2种图结构,利用访问序列图和跨序列图,实现了分别提取单个序列中数据项的信息和序列之间的关联信息.

3)在访问序列图中引入虚拟节点实现了对序列中的距离较远的数据项之间信息的学习,并将整个序列的全局信息聚合于该虚拟节点中.

4)将本文方法和最先进的基准模型进行比较,结果表明,在2个关键评估指标*Precision*@20(*P*@20)和Mean Reciprocal Rank at 20(*MRR*@20)上,Graph4Cache优于最先进的缓存预测模型.

1 相关工作

本节介绍了关于缓存替换策略的现有研究,并分析了这些方法和机制的特点.

1.1 基于规则的缓存策略

基于规则的缓存管理策略的研究主要集中在优化数据访问模式和增强存储效率方面. LRU-C[14]引入了 LRU-clean 指针来减少读取延迟，主要解决了传统数据库缓存管理中的 I/O 串行化问题. SpanDB[15]利用高速 SSD 技术提高了键值存储系统的性能，减少了对高端硬件的依赖. POCache[16]通过仅缓存奇偶块并采用可配置的滞后感知缓存算法(configurable straggler-aware caching algorithm, CSAC)来解决系统内节点性能不佳的问题，该算法提供了滞后者的检测和容忍功能，展示了在容错性方面的显著优势，尽管性能在很大程度上取决于纠删码操作的效率. DistCache[17]提出了一种新颖的分布式缓存机制，通过独立的哈希函数分区和双选查询路由策略实现了可验证的负载均衡. 传统的缓存替换策略主要依赖于数据项的历史访问模式来确定哪些数据应保留在缓存中、哪些应被淘汰. 这些策略主要依赖于数据访问的顺序或频率，忽视了数据项的全局或长期流行程度.

1.2 基于深度学习的缓存策略

深度学习在缓存预测领域的引入使缓存策略能够考虑全局数据热度信息，反映所有用户的数据项的频率或需求. 已有工作[2,18-19]引入了诸如 LSTM 等深度学习模型，以捕获关于数据热度的长期信息，用于预测潜在事件和内容需求，实现了内容感知型缓存策略，显著减少了后端网络拥塞和访问延迟. 然而，这些工作忽视了社交媒体数据的实时性，在处理动态变化的数据时难以保证预测的准确性. PA-Cache[3]利用多层递归神经网络和结构化深度神经网络来自适应地学习数据热度的时间变化，实现了高可伸缩性和计算效率. 一种基于深度强化学习的联合主动缓存放置和功率分配策略[20]提出通过对内容缓存和功率资源分配的智能决策，有效降低了延迟并提高了服务质量. 尽管利用深度学习模型构建缓存策略已经取得了有效的结果，但由于缓存环境的动态性和不确定性，基于深度学习的模型可能难以捕捉到所有关键的数据特征和模式，特别是在用户行为数据有限的情况下，这些模型不足以理解数据项之间的复杂转换和跨序列的全局信息.

2 模型概述

在本节中，我们首先介绍缓存预取的概念和定义，然后展示如何图形化地建模缓存访问序列. 接下来，介绍生成访问序列中项的图嵌入方法，并最终根据这些嵌入生成缓存预取项. 图 1 展示了 Graph4Cache 的工作流程.

该模型分为 3 个部分：1)访问数据序列建模；2)跨序列数据建模；3)缓存预取生成.

在访问数据序列建模部分，我们为每一个访问序列都构建一个带有虚拟节点的访问序列图(access sequence graph, ASGraph)，其中，棕色节点表示虚拟节点，绿色节点表示用户访问序列中被访问过的数据对象，黑色实线表示被连续访问的数据对象之间通过有向边相连，蓝色虚线表示虚拟节点与所有节点的双向连接. 虚拟节点可以学习到图中其他节点的信息，而其他所有节点也都能通过虚拟节点学习到非相邻节点的信息. 通过图嵌入层为 ASGraph 中的每个节点生成嵌入表示；在此基础之上进行跨序列数据建模. 使用虚拟节点的表示，构建能够体现序列相关性的跨序列图(cross sequence graph, CSGraph)，并使用 GCN 学习 CSGraph 中的虚拟节点的表示. 最

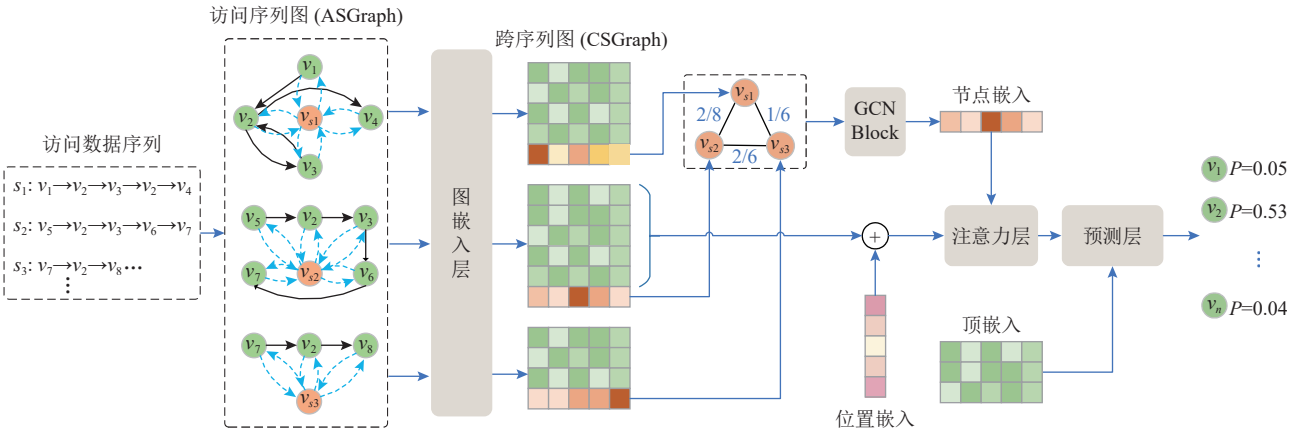


Fig. 1 The workflow of Graph4Cache

图 1 Graph4Cache 工作流程

后基于注意力机制生成缓存预取数据，结合从 ASGraph 学习到的序列内部特征和从 CSGraph 学习到的跨序列特征，并引入待预测序列中节点的位置编码，生成缓存预取的表示.

2.1 问题描述

缓存预取的目的是通过预测将要访问的数据对象并将其预加载到存储介质中来提高访问效率. 为了实现这一目标，我们将一段时间内缓存用户的访问历史抽象为一个序列，并基于这些访问历史进行预测. 具体来说，令 $V=\{v_1,v_2,\cdots,v_N\}$表示系统中的数据对象集，其中每个数据对象都是唯一的，N 表示唯一数据对象的总数. 给定一个序列 $s=\{v_1,v_2,\cdots,v_n\}$，其中 v_i 表示序列中的第 i 个数据项，目标是预测将被访问的 v_{n+1}. 表 1 总结了本文用到的符号.

Table 1 Description of Important Notions

表 1 重要符号描述

符号	描述
V	缓存数据对象的集合
s	访问序列
G_s	访问序列图 ASGraph
G_g	跨序列图 CSGraph
$\boldsymbol{A}_{in}$	入度矩阵
$\boldsymbol{A}_{out}$	出度矩阵
$\boldsymbol{h}_i^t$	时刻 t 节点 i 的嵌入
$\boldsymbol{x}_s^t$	时刻 t 序列 s 的虚拟节点的嵌入
$\boldsymbol{S}$	用户意图嵌入

2.2 数据访问序列的图模型

为了充分学习用户的历史访问序列中数据项之间的转换关系，我们根据每个序列的自然顺序将其建模为有向加权图的形式. 具体来说，该图中的每个节点表示一个数据项，有向边表示数据对象之间的顺序访问关系. 此外添加了一个虚拟节点，连接到有向加权图中所有其他节点，以在图中的非相邻项之间传播信息. 每个访问序列 $s=\{v_1,v_2,\cdots,v_n\}$表示为一个访问序列图 ASGraph: $G_s=(V_s,E_s)$，其中 $V_s=V_s'\cup\{v_s\}$ 表示 G_s 中的顶点集. $V_s'=\{v_1,v_2,\cdots,v_m\}$对应于 s 中的 m 个唯一数据对象，由于可能有项目的重复，$m\leqslant n$；v_s 表示新添加的虚拟节点，E_s是边的集合.

对于序列 s 中连续访问的任意 2 个数据项 $v_i,v_j\in V_s$，存在一个有向边 $(v_i,v_j)\in E_s$，表示用户在 s 中顺序访问了 v_i和 v_j. 考虑到这种连续访问可能在 s 中发生多次，可以构建对应的入度矩阵 $\boldsymbol{A}_{in}$和出度矩阵 $\boldsymbol{A}_{out}$来表示节点之间的有向边及其权重，其中权重计算为边的频率除以边的起始节点的出度. 以序列 $s=\{v_3,v_1,v_3,v_7,v_6\}$为例，构建其入度矩阵和出度矩阵，如图 2 所示.

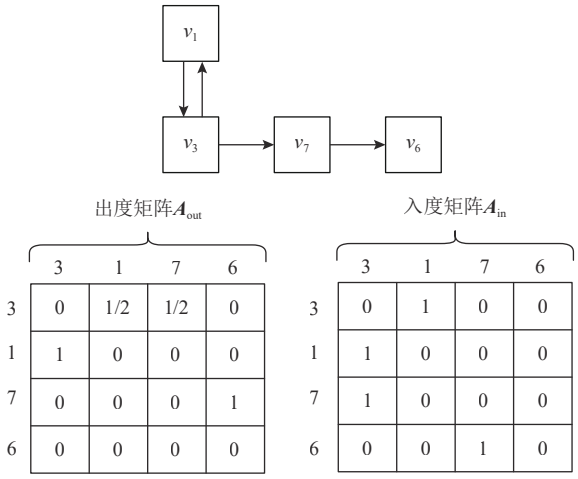


Fig. 2 Illustration of incoming and outgoing matrices

图 2 入度矩阵和出度矩阵示意图

为了促进非相邻节点之间的信息传播，在 ASGraph 中添加虚拟节点 v_s，将其连接到所有其他节点，即在 v_s 和 G_s 中的每个节点之间添加双向边. 一方面对于任意 $v_i\in V_s'$，通过有向边 $(v_i,v_s)\in E_s$来更新 v_s 的表示，使其能够从图中所有其他节点学习信息. 另一方面通过有向边 $(v_s,v_i)\in E_s$更新 v_i 的表示，以 v_s 作为桥梁，允许 v_i 从非相邻节点学习信息. 与传统的构建序列图的方式相比，添加虚拟节点有助于在远距离节点之间进行消息传递，并有助于学习图的全局信息.

通过在 ASGraph 中引入虚拟节点，我们学习到了访问序列内部数据对象的转换关系，在此基础上，我们使用虚拟结点构建跨序列图 CSGraph: $G_g=\{V_g,E_g\}$来学习访问序列之间的相关信息. 具体地，$V_g=\{v_{s_1},v_{s_2},\cdots,v_{s_l}\}$，其中 v_{s_i} 表示序列 s_i 在 ASGraphG_{s_i} 的虚拟节点，l 表示序列的总数. 如果G_s和G_{s_j}共享节点，则存在一条边 $(x_{s_i},x_{s_j})\in E_g$. 我们为每条边分配一个权重$w_{i,j}$，计算公式为

$$w_{i,j}=\frac{\left|V_{s_i}'\cap V_{s_j}'\right|}{\left|V_{s_i}'\cup V_{s_j}'\right|}. \tag{1}$$

由此形成 CSGraph 的邻接矩阵 $\boldsymbol{A}\in\mathbb{R}^{l\times l}$.

2.3 数据项的图嵌入表示

1)访问序列图学习

对于任意数据对象 $v\in V$都被嵌入到一个统一的向量空间中，向量 $\boldsymbol{h}_i\in\mathbb{R}^d$表示数据对象 v_i的潜在向量表示，其中 d 是向量的维度. v_i的初始嵌入表示为 $\boldsymbol{h}_i^0$，可以通过将每个数据对象的 one-hot 编码与可训练的

权重矩阵相乘来获得. 对于虚拟节点v_s, 其初始嵌入$\boldsymbol{x}_s^0$可通过对序列中其他节点的嵌入进行平均池化来获得, 即:

$$\boldsymbol{x}_s^0=\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{h}_i^0. \tag{2}$$

针对普通节点的更新, 为了有效利用 ASGraph 中边的方向和权重信息, 我们采用门控图神经网络[21](gated graph neural network, GGNN)来学习图中节点的嵌入表示. GGNN 将递归神经网络的概念融入 GNN 中, 实现了顺序和迭代的更新过程. 与传统的 GNN 相比, GGNN 能够有选择地结合邻近节点的相关信息, 实现灵活的信息传递. 其门控机制允许自适应学习和选择性信息更新, 使节点能够在处理和传递重要信息同时过滤噪声, GGNN 还可以捕获图的时间动态和长程依赖关系. GGNN 处理图的过程分为 2 个步骤: 消息传递和消息聚合.

在消息传递过程中, 每个节点都会分别在出度方向和入度方向聚合来自邻居节点的信息, 计算表达式为:

$$\boldsymbol{\alpha}_{\mathrm{out}_i}^t=\boldsymbol{A}_{\mathrm{out}_i}\left(\boldsymbol{h}_1^{t-1},\boldsymbol{h}_2^{t-1},\cdots,\boldsymbol{h}_n^{t-1}\right)^{\mathrm{T}}\boldsymbol{W}_{\mathrm{out}}, \tag{3}$$

$$\boldsymbol{\alpha}_{\mathrm{in}_i}^t=\boldsymbol{A}_{\mathrm{in}_i}\left(\boldsymbol{h}_1^{t-1},\boldsymbol{h}_2^{t-1},\cdots,\boldsymbol{h}_n^{t-1}\right)^{\mathrm{T}}\boldsymbol{W}_{\mathrm{in}}, \tag{4}$$

$$\boldsymbol{\alpha}_i^t=Concat\left(\boldsymbol{a}_{\mathrm{in}_i}^t,\boldsymbol{a}_{\mathrm{out}_i}^t\right). \tag{5}$$

以 GGNN 的第 t 层为例, 其输入来自前一层的每个节点的嵌入$\left(\boldsymbol{h}_1^{t-1},\boldsymbol{h}_2^{t-1},\cdots,\boldsymbol{h}_n^{t-1}\right)$, $\boldsymbol{A}_{\mathrm{in}_i}$和$\boldsymbol{A}_{\mathrm{out}_i}$分别提取 ASGraph 的入度矩阵和出度矩阵的第 i 行来表示节点v_i和其他节点之间的连接关系和相应的权重信息. 由于 ASGraph 是双向的, 因此考虑了 2 个参数矩阵$\boldsymbol{W}_{\mathrm{in}}$和$\boldsymbol{W}_{\mathrm{out}}$用于编码不同方向的输入, 反映双向消息传播, 最后再拼接来自 2 个方向的消息$\boldsymbol{\alpha}_{\mathrm{in}_i}^t$和$\boldsymbol{\alpha}_{\mathrm{out}_i}^t$.

一旦节点之间的消息传递被定义, 使用门控循环单元(GRU)来合并来自其他节点的消息并更新每个节点的隐藏状态:

$$\boldsymbol{z}_i^t=\sigma\left(\boldsymbol{W}_z\boldsymbol{\alpha}_i^t+\boldsymbol{U}_z\boldsymbol{h}_i^{t-1}\right), \tag{6}$$

$$\boldsymbol{r}_i^t=\sigma\left(\boldsymbol{W}_r\boldsymbol{\alpha}_i^t+\boldsymbol{U}_r\boldsymbol{h}_i^{t-1}\right), \tag{7}$$

$$\tilde{\boldsymbol{h}}_i^t=\tanh\left(\boldsymbol{W}_o\boldsymbol{a}_i^t+\boldsymbol{U}_o\left(\boldsymbol{r}_i^t\odot\boldsymbol{h}_i^{t-1}\right)\right), \tag{8}$$

$$\hat{\boldsymbol{h}}_i^t=\left(1-\boldsymbol{z}_i^t\right)\odot\boldsymbol{h}_i^{t-1}+\boldsymbol{z}_i^t\odot\tilde{\boldsymbol{h}}_i^t, \tag{9}$$

其中, $\boldsymbol{z}_i$和$\boldsymbol{r}_i$分别表示更新门和重置门, 控制了信息从过去到当前的迭代. 在这个过程中, 每个节点还从虚拟节点接收信息. 在 ASGraph 中, 虚拟节点代表了序列的整体信息, 对于每个普通节点, 计算它与虚拟节点的相似度:

$$\boldsymbol{\alpha}_i^t=\frac{\left(\boldsymbol{W}_{q_1}\hat{\boldsymbol{h}}_i^t\right)^{\mathrm{T}}\boldsymbol{W}_{k_1}\boldsymbol{x}_s^{t-1}}{\sqrt{d}}, \tag{10}$$

其中, $\boldsymbol{W}_{q_1}$和$\boldsymbol{W}_{k_1}$是可训练参数. 最后使用门控网络来有选择地整合来自邻居节点和虚拟节点的信息:

$$\boldsymbol{h}_i^t=\left(1-\boldsymbol{\alpha}_i^t\right)\hat{\boldsymbol{h}}_i^t+\boldsymbol{\alpha}_i^t\boldsymbol{x}_s^{t-1}. \tag{11}$$

针对虚拟节点的更新, 我们首先引入自注意力机制来计算每个普通节点对虚拟节点的重要性, 即:

$$\beta=softmax\left(\frac{\left(\boldsymbol{W}_{k_2}\boldsymbol{h}^t\right)^{\mathrm{T}}\boldsymbol{W}_{q_2}\boldsymbol{x}_s^{t-1}}{\sqrt{d}}\right), \tag{12}$$

其中, $\boldsymbol{W}_{k_2}$和$\boldsymbol{W}_{q_2}$分别是用于转换普通节点和虚拟节点的可训练参数. 在获取每个普通节点的权重后, 通过线性组合得到虚拟节点的表示:

$$\boldsymbol{x}_s^t=\beta\boldsymbol{h}^t. \tag{13}$$

经过 T 次传播后, 可以得到普通节点的嵌入$\boldsymbol{h}^*=\left\{\boldsymbol{h}_1^*,\boldsymbol{h}_2^*,\cdots,\boldsymbol{h}_n^*\right\}$以及 ASGraph 中虚拟节点的嵌入$\boldsymbol{x}_s^*$. 每个节点的嵌入不仅包含自身的特征, 还汇聚了来自不同方向的*阶邻居节点的信息. 最后使用 Highway 网络解决堆叠多层 GNN 可能导致的过拟合, 允许从节点的初始嵌入和最终嵌入中选择性地检索信息, 从而得到普通节点的最终表示:

$$\boldsymbol{h}^f=\boldsymbol{g}\odot\boldsymbol{h}^0+(1-\boldsymbol{g})\odot\boldsymbol{h}^*, \tag{14}$$

其中, $\boldsymbol{g}$ 由$\boldsymbol{h}^0$和$\boldsymbol{h}^*$决定:

$$\boldsymbol{g}=\sigma\left(\boldsymbol{W}_{\mathrm{g}}\left[\boldsymbol{h}^0\|\boldsymbol{h}^*\right]\right), \tag{15}$$

其中, ||表示连接操作, $\boldsymbol{W}_{\mathrm{g}}\in\mathbb{R}^{d\times 2d}$是一个可训练参数, 用于将来自$\mathbb{R}^{2d}$的向量转换为$\mathbb{R}^d$, σ是激活函数. 通过 *Highway* 网络, 可以得到普通节点的最终表示$\boldsymbol{h}^f$, 虚拟节点表示为$\boldsymbol{x}_s^*$.

2)跨序列图学习

基于学习到的虚拟节点表示为$x=\left\{\boldsymbol{x}_{s_1},\boldsymbol{x}_{s_2},\cdots,\boldsymbol{x}_{s_l}\right\}$, 可将构建的 CSGraph 输入到 GCN 中, 以学习每个序列的全局特征, 并捕捉序列之间的潜在连接. 根据 CSGraph 的定义, 存在一个邻接矩阵$\boldsymbol{A}\in\mathbb{R}^{l\times l}$, 其中$A_{p,q}=w_{p,q}$表示序列$s_q$, s_p之间共享节点的数量. 定义$\hat{\boldsymbol{A}}=\boldsymbol{A}+\boldsymbol{I}$, 其中 $\boldsymbol{I}$ 是单位矩阵, 并定义度矩阵$\hat{\boldsymbol{D}}\in\mathbb{R}^{l\times l}$, 其中$\hat{D}_{p,p}=\sum_{q=1}^{l}\hat{A}_{p,q}$. 基于这些定义, 可以推导得:

$$\boldsymbol{x}^{k+1}=\hat{\boldsymbol{D}}^{-1}\hat{\boldsymbol{A}}\boldsymbol{x}^k\boldsymbol{W}_c^k, \tag{16}$$

其中, $\boldsymbol{W}_c^k$是第 k 层的可训练矩阵, 每个节点的卷积收集其邻近节点的消息, 并将其与自身的表示融合, 以便每个虚拟节点从其他序列中学习信息. 经过 K 层卷积层后, 我们得到每个节点的最终表示$\boldsymbol{x}^K$. 为了防止过拟合并提高模型的泛化能力, 计算每层输出的

平均值:

$$\boldsymbol{X}=\frac{1}{K+1}\sum_{k=0}^{K}\boldsymbol{x}^{k}. \tag{17}$$

由此得出每个序列中虚拟节点的最终表示$\boldsymbol{X}=(\boldsymbol{X}_{s_1},\boldsymbol{X}_{s_2},\ldots,\boldsymbol{X}_{s_l})$.

2.4 预取结果生成

对于每个要预测的序列$s=\{v_1,v_2,\cdots,v_n\}$,从ASGraph中学习其局部特征$\boldsymbol{h}$,从CSGraph中学习其全局特征$\boldsymbol{X}_s$,则可将序列的这2个特征结合起来生成缓存预取结果.

由于有向加权图并不能反映图中节点之间的先后访问顺序信息,因此我们引入了一个可学习的位置嵌入矩阵$\boldsymbol{P}=(\boldsymbol{p}_1,\boldsymbol{p}_2,\cdots,\boldsymbol{p}_n)$,其中$\boldsymbol{p}_i\in\mathbb{R}^d$是与序列中$v_i$对应的位置向量.可将位置信息添加到节点表示中,得到$\boldsymbol{h}^p=\boldsymbol{h}+\boldsymbol{P}$.

我们从序列的局部特征中选择最后一项$\boldsymbol{h}_n^p$作为用户最新意图的表示.考虑到不同项目对最终预测的影响不同,可将跨序列的全局特征进行组合,并使用注意力机制来计算每个数据对象的影响因子,计算为:

$$\gamma_i=\boldsymbol{W}^{*}\sigma\left(\boldsymbol{W}_h\boldsymbol{h}_i^p+\boldsymbol{W}_x\boldsymbol{x}_s+\boldsymbol{W}_n\boldsymbol{h}_n^p+\boldsymbol{b}\right), \tag{18}$$

以上参数$\boldsymbol{W}$、$\boldsymbol{W}_h$、$\boldsymbol{W}_x$、$\boldsymbol{W}_n$以及$\boldsymbol{b}$是可训练的.以此计算每个项目嵌入的加权和以获取长期意图:

$$\boldsymbol{s}_{\mathrm{g}}=\sum_{i=1}^{n}\gamma_i\boldsymbol{h}_i^p. \tag{19}$$

最后将长期意图和最新意图连接起来,得到用户意图的最终表示:

$$\boldsymbol{S}=\boldsymbol{W}_s\left[\boldsymbol{s}_{\mathrm{g}}\|\boldsymbol{h}_n^p\right], \tag{20}$$

其中,$\boldsymbol{W}_s$是可训练参数,用户意图的最终表示由当前序列中涉及的所有项目构建,每个项目的贡献不仅由其位置决定,还受到序列的全局特征的影响.

根据获取的用户意图以及每个候选数据对象的初始嵌入$\boldsymbol{h}_i^0$,可以使用softmax获得一个概率分布$\hat{\boldsymbol{y}}=(\hat{\boldsymbol{y}}_1,\hat{\boldsymbol{y}}_2,\cdots,\hat{\boldsymbol{y}}_N)$,即:

$$\hat{\boldsymbol{y}}_i=softmax\left(\boldsymbol{S}^{*}\boldsymbol{h}_i^0\right), \tag{21}$$

其中,$\hat{\boldsymbol{y}}_i\in\hat{\boldsymbol{y}}$表示在当前序列中$\boldsymbol{y}_i$将被下一次访问的概率.最终的损失函数定义为预测$\hat{\boldsymbol{y}}$和真实标签$\boldsymbol{y}$之间的交叉熵:

$$\mathcal{L}(\hat{\boldsymbol{y}})=-\sum_{i=1}^{N}\boldsymbol{y}_i\log(\hat{\boldsymbol{y}}_i)+(1-\boldsymbol{y}_i)\log(1-\hat{\boldsymbol{y}}_i), \tag{22}$$

其中,$\boldsymbol{y}$表示真实标签的one-hot编码向量.

3 实验设置

为验证本文提出的Graph4Cache的有效性,本文主要从预测准确性和缓存命中率2个方面设计并开展相关实验.本节主要介绍实验的基本设置,包括采用的数据集、模型评估指标与标准、基准比较模型和模型参数配置等.

3.1 数据集

我们一共选取了3种主流公开的基准数据集,包括Diginetica, Tmall和Nowplaying. Diginetica源于国际数据挖掘竞赛ACM CIKM Cup(2016),其包含从电子商务搜索引擎日志中提取的匿名用户会话信息. Tmall是国际人工智能联合会议与阿里巴巴公司合办的IJCAI-15竞赛中采用的数据集,它涵盖了天猫在线购物平台的匿名用户购物日志. Nowplaying[22]是从Twitter上收集到的4 900万条用户的音乐收听行为数据集,包括用户名、时间戳、歌曲名等字段信息.

我们参考先前工作[10,23]的做法对3个数据集进行预处理.具体来说,我们过滤掉数据项出现次数小于5的会话,并将最后一周的会话(也就是最新的会话)作为测试数据,其余的历史会话用于训练.之后对于数据集中的每个会话$\boldsymbol{S}=(s_1,s_2,\cdots,s_n)$,通过序列分裂的方法对数据集进行标注和扩增,生成多个带有对应标签的标注序列,即$([s_1],s_2),([s_1,s_2],s_3),\cdots,([s_1,s_2,\cdots,s_{n-1}],s_n)$.预处理后的数据集的统计数据如表2所示.

Table 2 Statistics of the Datasets

表2 数据集统计

	Diginetica	Tmall	Nowplaying
#click	982 961	818 479	1 367 963
#train	719 470	351 268	825 304
#test	60 858	25 898	89 824
#items	43 097	40 728	60 417
Avg. len.	5.12	6.69	7.42

3.2 评估指标

借鉴会话型推荐(session-based recommendation, SBR)领域的先前工作[23-24],我们使用*P*@20和*MRR*@20评估所提出的模型.*P*@20广泛用于衡量预测准确性,计算模型生成的推荐列表中前20个位置中出现的正确推荐项的比例.另一方面,*MRR*@20考虑模型生成

的推荐列表中目标项的平均倒数排名，并将倒数排名设置为 20 个位置之外的排名为 0. 较高的 *MRR* 值表示推荐列表中真实值的排名较高.

在模拟缓存的情况下，我们采用缓存命中率作为评估指标，它被定义为缓存命中总数与总内存访问次数的比率.

3.3 基准模型

在本节中，我们选取了多种推荐和预测领域的模型，包括 3 种传统方法，即 POP, Item-KNN[25], FPMC[26]; 基于 RNN 的方法，即 GRU4Rec[9], NARM[26], CSRM[27]; 基于 Attention 的方法，即 STAMP[23]; 基于 GNN 的方法，即 SR-GNN[10], GCE-GNN[13], SGNN-HN[12], DHCN[14], COTREC[28], MAE-GNN[29], GS-GNN[30]. 以上模型的基本介绍为：

1）POP. 推荐在训练集中最频繁出现的前 N 个项.

2）Item-KNN. 根据会话中项目之间的余弦相似性进行推荐.

3）FPMC. 结合矩阵分解和一阶马尔科夫链来捕捉序列信息和用户偏好，但在计算推荐分数时忽略了用户的潜在表示.

4）GRU4Rec. 一种基于 RNN 的模型，使用 GRU 来建模用户序列.

5）NARM. 在 RNN 的基础上引入注意力机制来改善 GRU4Rec 的表现.

6）CSRM. 在 NARM 的基础上引入了邻近会话作为辅助信息，并通过并行内存模块对当前会话进行建模.

7）STAMP. 利用注意力机制来捕捉一般兴趣，并将当前会话中的最后一个项视为最近的兴趣进行预测.

8）SR-GNN. 使用门控图神经网络获取项嵌入，并使用注意力机制生成会话表示.

9）GCE-GNN. 构建会话图和全局图以学习会话级别和全局级别的项嵌入，并使用注意力机制合并这 2 个嵌入表示.

10）SGNN-HN. 设计星形神经网络来捕捉会话中远距离项之间的转换关系，并使用 Highway 网络来防止过拟合.

11）DHCN. 设计了双通道超图卷积网络来捕捉项之间的高阶关系，并将自监督学习整合到网络训练过程中.

12）COTREC. 构建了会话内部和外部连接的 2 个视图，并通过对比学习实现自监督协同训练，解决了数据稀疏性问题.

13）MAE-GNN. 结合双门控图神经网络和多头注意力机制，选择重要的节点信息，从多个维度提取用户的兴趣和偏好.

14）GS-GNN. 将整个数据集建模为全局图，并使用图注意力网络学习项目的全局表示，并通过图神经网络整合项目的全局特征和局部特征.

3.4 模型配置

我们采用一层 GGNN 来获取节点的嵌入表示，并采用 3 层图卷积网络来学习 CSGraph 的特征. 与之前的研究[23-24] 一致，验证集配置为训练集的随机 10% 子集，批量大小设置为 100，节点嵌入的维度为 256. 我们使用 Adam 优化器，初始学习率为 0.001，在每 3 个周期后衰减为 0.1. L2 正则化设置为 1E–5 以防止过拟合，并且将 3 个数据集的比例因子设置为 12.

4 实验结果及分析

本节设计了一系列实验来验证模型的有效性，主要可分为 2 个部分：序列推荐实验和缓存模拟实验. 序列推荐实验主要验证模型在预测方面的准确性，并与传统的序列推荐模型进行比较. 另一方面，缓存模拟实验旨在验证模型在真实缓存场景中的性能，通过将其与传统缓存替换策略集成以提高缓存命中率.

4.1 结果对比

我们在每个数据集上都运行模型 5 次，并记录了实验结果的平均值. 表 3 展示了 11 个基准模型和我们提出的模型在 3 个真实数据集上的实验结果. 观察表 3 可以发现，Graph4Cache 在大部分指标上都优于其他基线模型，相较于之前的最佳结果，Graph4Cache 带来了 0.29%~5.45% 的性能提升. 而在 Nowplaying 数据集的 *P*@20 指标和 Tmall 数据集的 *MRR*@20 指标上，尽管没有取得最优，但也都逼近最佳结果.

在传统方法中，由于只推荐前 N 个频繁出现的项目，POP 的性能最差. 与 POP 相比，FPMC 通过利用一阶马尔可夫链和矩阵分解，在 3 个数据集上表现相对较好. Item-KNN 在 Diginetica 和 Nowplaying 数据集上的表现是传统方法中最好的. 与传统方法相比，基于深度神经网络的方法由于考虑了会话中的顺序依赖性，通常表现出更好的会话推荐性能. 其中，GRU4Rec 作为第一个使用 RNN 进行会话推荐的方法，在 Diginetica 和 Nowplaying 数据集上的表现不如基于相似性进行推荐的 Item-KNN，原因可能是 RNN 专门设计用于序列建模，而会话推荐并不仅仅是一

Table 3 Performance Comparison of All Methods on Three Datasets

表 3 所有方法在 3 个数据集上的性能对比

模型	Diginetica		Nowplaying		Tmall	
	P@20	*MRR*@20	*P*@20	*MRR*@20	*P*@20	*MRR*@20
POP	1.18	0.28	2.28	0.86	2.00	0.90
Item-KNN	35.75	11.57	15.94	4.91	9.15	3.31
FPMC	26.53	6.95	7.36	2.82	16.06	7.32
GRU4Rec	29.45	8.33	7.92	4.48	10.93	5.89
NARM	49.70	16.17	18.59	6.93	23.30	10.70
CSRM	50.55	16.38	18.14	6.42	29.46	13.96
STAMP	46.62	15.13	17.66	6.88	26.47	13.36
SR-GNN	51.26	17.78	18.87	7.47	27.57	13.72
GCE-GNN	54.22	19.04	22.37	8.40	33.42	15.42
SGNN-HN	54.57	19.02	19.97	7.74	35.43	16.59
DHCN	53.66	18.51	**23.03**	8.18	30.43	14.26
COTREC	54.18	19.07			36.35	**18.04**
MAE-GNN	52.50	17.99			33.44	15.37
GS-GNN	53.08	18.91			32.65	15.70
Graph4Cache（本文）	**54.73**	**19.15**	21.41	**8.43**	**38.33**	17.95
提升率/%	0.29	0.42		0.36	5.45	

注：其中每列中的最佳结果用粗体标出，次佳结果用下划线标出.

个序列建模任务，因为用户兴趣可能会在一个会话序列中发生变化. 而引入了注意力的神经网络方法 NARM 和 STAMP 明显优于 GRU4Rec. NARM 将 RNN 与注意力机制结合起来，使用 RNN 的最后一个隐藏状态作为用户的主要偏好. 这个结果表明，直接用 RNN 对会话序列进行编码可能不足以用于会话推荐，因为 RNN 只能模拟会话中相邻项目之间的单向转换. 我们还观察到，完全基于注意力的方法 STAMP 在 Tmall 数据集上的性能优于 NARM. 这个结果验证了在会话编码中给不同项目分配不同注意力权重的有效性. 与 RNN 相比，注意力机制也似乎是解决会话推荐问题的一个可行选项. CSRM 在 Diginetica 和 Tmall 上的表现优于 NARM 和 STAMP，这说明引入其他会话信息的有效性. 但 CSRM 将其他会话视为一个整体，并未区分其他会话中的相关项目和无关项目.

在所有基准方法中，基于 GNN 的方法在 Diginetica 和 Nowplaying 数据集上表现更好. SR-GNN 将每个会话序列建模为一个子图，并使用 GNN 对项目进行编码，然后通过注意力机制集成会话信息进行预测，证明了 GNN 在会话推荐中可以提取用户在会话中表现出的偏好. 这表明，图模型比序列建模、RNN 或注意力建模更适用于会话推荐.

之后的 GNN 方法各有优缺点. GCE-GNN 和 GS-GNN 都整合了项目的全局特征和局部特征，但 GCE-GNN 的全局图构建仍然侧重于项目，未能充分反映跨会话关系，而 GS-GNN 为整个数据集构建全局图，引入了过多噪声，且带来了较高的计算成本. SGNN-HN 引入星形节点来捕捉会话中远距离的项目关系，但只从单个会话获取信息. MAE-GNN 使用注意力机制，从多个维度提取用户的兴趣和爱好，但也只考虑了单个会话序列，忽略了跨会话信息. DHCN 构建超图和线图来捕捉项目之间的高阶关系，但构图方式相对复杂. COTREC 通过对比学习实现了自监督，结合协同训练实现了基于会话的数据增强，在数据稀疏时更为有效. 这些方法在 SR-GNN 基础上都略有改进. 本文方法综合了先前 GNN 的优点，并提供了一个相对简单的图构建方法. 它在多个数据集指标上取得了最佳或接近最佳的性能，相较于之前的最佳结果，取得了 0.29%~5.45% 的性能提升.

4.2 消融实验

为了验证我们提出的 AS-GNN 模型的有效性，我们在 3 个数据集上进行了比较实验，分别移除和保留了该模块，并使用 *P*@20 和 *MRR*@20 作为评估指标. 比较模型为：

1）w/o global. 没有构建和传播全局图的特征，只实现了 ASGraph 的构建和特征传播. 星形节点直接用作会话的嵌入表示.

2）1/2/3 跳. 在 CSGraph 的构建和特征传播的基础上，将传播层数设置为 1 层，2 层，3 层，分别进行评估.

表 4 对不同模型进行了比较，清晰地展示了 AS-GNN 模块的有效性. 与 w/o global（没有 CSGraph 的模型）相比，CSGraph 的构建和特征传播导致了模型整体性能的显著改进. 尤其是在 3 层传播时，CS-GNN 模块能为模型的 2 个指标带来 4.56%~8.85% 的性能

Table 4 Performance Comparison of Models in Ablation Study

表 4 消融实验中模型性能比较

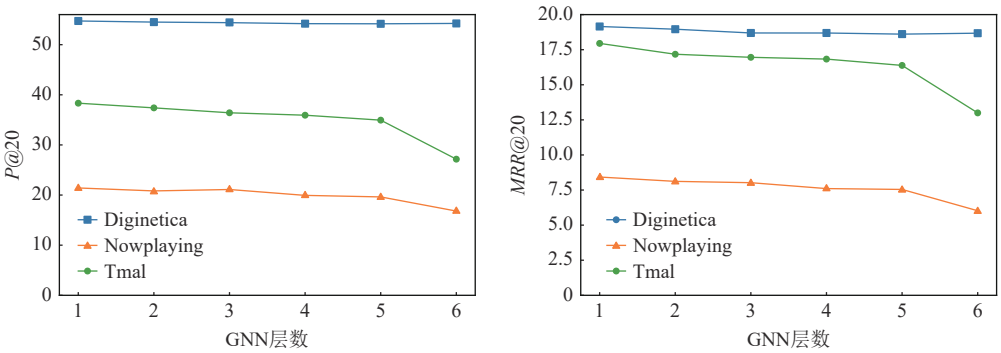
模型	Diginetica		Nowplaying		Tmall	
	P@20	*MRR*@20	*P*@20	*MRR*@20	*P*@20	*MRR*@20
w/o global	52.341	18.020	19.969	7.742	35.431	16.594
1-跳	54.768	19.114	21.697	8.453	38.157	17.893
2-跳	54.769	19.112	21.317	8.330	38.188	17.944
3-跳	54.727	19.152	21.414	8.427	38.327	17.948
提升率/%	4.560	6.280	7.240	8.850	8.170	8.450

提升，这表明 CS-GNN 模块能增加模型从其他会话中挖掘跨会话相关信息的能力，从而帮助模型做出更准确的预测. 在不同传播层数的全局特征传播下，观察到在 3 层传播时性能略有提高. 这可能表明更广泛的全局信息聚合可以获得更有效的洞察力.

4.3 GNN 网络深度影响分析

在 GNN 领域，网络层数是影响性能的关键超参数. 为了研究 CS-GNN 模块中层数的影响，我们将 GNN 层数设置为 1~6，并展示在 3 个数据集上 *P*@20 和 *MRR*@20 的性能. 如图 3 所示，图 3(a)和图 3(b)分别展示了模型在不同数据集和不同层数下的性能.

随着层数的增加，模型在 Tmall 和 Nowplaying 数据集上的性能下降，可能是因为虚拟结点的引入已经使得 ASGraph 中的每个节点都能在 1 跳或 2 跳的距离内学习到图中其他任意一个节点的信息，因此一旦增加层数反而容易引起过拟合. 同时注意到，模型在 Diginetica 数据集上的性能对层数的变化不太敏感，这可能是由于 Diginetica 数据集有更广泛的数据分布和更多样的样本，防止了模型在训练数据集中对特定实例的过度依赖.



(a) *P*@20 (b) *MRR*@20

Fig. 3 Performance of the model with different numbers of GNN layers

图 3 不同数量 GNN 层的模型性能表现

以上实验结果表明，使用 1 层 GNN 学习 ASGraph 和 3 层 GNN 学习 CSGraph 可以使模型取得最佳性能.

4.4 缓存模拟实验的研究及分析

为了验证模型在真实缓存场景中的性能，我们进行了缓存模拟实验，将 Graph4Cache 与不同的传统缓存替换算法相结合，并分析了在冷启动场景下和多用户并发场景下，缓存容量对模型命中率的影响.

使用的传统缓存替换算法有：

1)LRU(least recently used)算法. 每次替换最久未被访问的数据，假设最近未被访问的数据最不可能在将来被访问.

2)LFU(least frequently used)算法. 根据数据项被访问的频率，淘汰访问次数最少的数据项，假设最近访问频率最低的数据项最不可能在将来被访问.

3)FIFO(first in first out)算法. 替换最先进入缓存的数据，假设最早进入的数据最不可能在将来被访问.

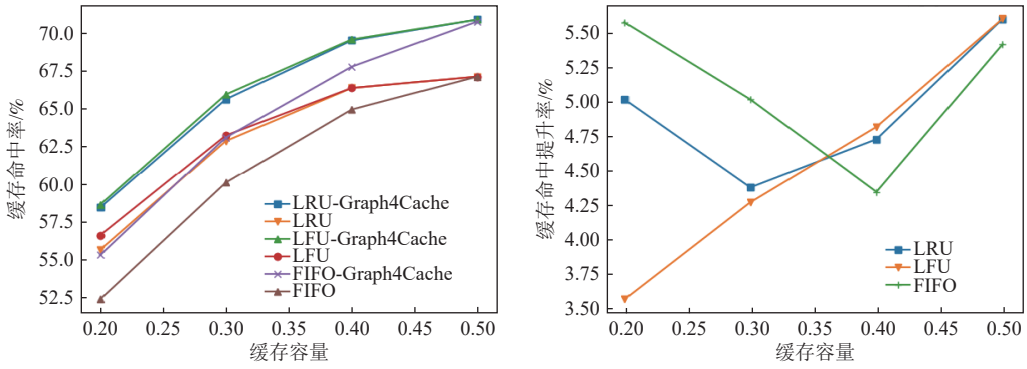
4.4.1 冷启动场景

在冷启动场景中，将 Graph4Cache 与传统的缓存替换算法相结合，观察 Graph4Cache 对缓存命中率的影响. 每个模型和算法的命中率随缓存容量的变化如图 4(a)所示，Graph4Cache 对命中率的提升效果如图 4(b)所示. 图 4(a)展示了 Graph4Cache 与 LRU, LFU 和 FIFO 相结合前后的缓存命中率的变化情况. 所有缓存算法的命中率随着缓存容量的增加而提升. 在传统的缓存替换策略中，LFU 和 LRU 的表现优于 FIFO. 值得注意的是，将 Graph4Cache 与 LFU 和 LRU 算法相结合的模型显示出比其他模型显著更高的缓存命中率. 图 4(b)展示了 Graph4Cache 对 LRU, LFU 和 FIFO 的改进效果. 在缓存容量相对较小时，Graph4Cache 对 FIFO 和 LRU 的性能改进要比 LFU 的显著；而在缓存容量较大时下，它对 LRU, LFU 和 FIFO 都表现出显著的改进效果.

上述实验结果表明，在冷启动场景中，Graph4Cache 能够根据有限的用户访问序列历史预测下一个项目，并将数据预取到缓存中. 这有效提高了缓存命中率，在缓存容量有限时尤其有意义.

4.4.2 多用户并发场景

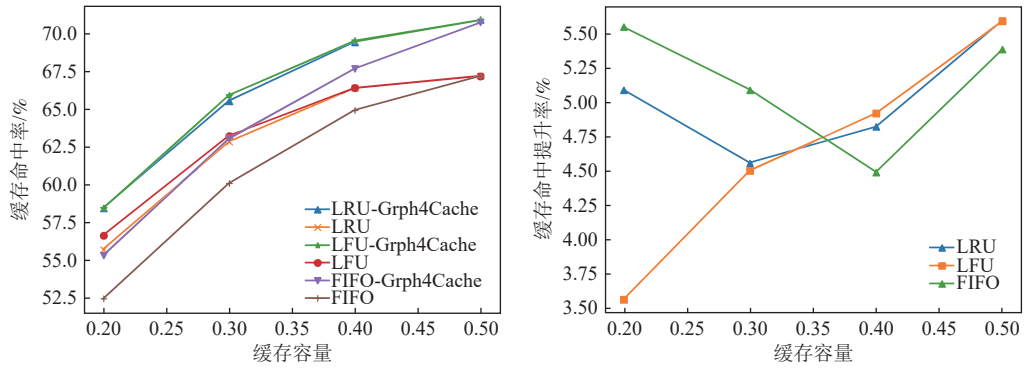
当系统中存在 100 个用户同时发出缓存访问请求时，Graph4Cache 对传统缓存替换算法的改进效果如图 5 所示. 其中各个模型和算法的命中率随缓存容量的变化如图 5(a)所示，Graph4Cache 对命中率的改进效果如图 5(b)所示.



(a) 命中率 (b) 命中率提升度

Fig. 4 Performance of Graph4Cache in cold start scenarios

图 4 Graph4Cache 在冷启动场景下的表现



(a) 命中率 (b) 命中率提升度

Fig. 5 Performance of Graph4Cache in multi-user concurrent scenarios

图 5 Graph4Cache 在多用户并发场景下的表现

在图 5(a)中，我们观察到所有算法的命中率随着缓存容量的增加而提升. 其中，Graph4Cache 分别与 LRU 和 LFU 算法相结合的模型显示出最高的缓存命中率，而将 Graph4Cache 与 FIFO 算法相结合的模型显示出次优的性能. 在图 5(b)中，观察到 Graph4Cache 能够显著提高 LRU 和 FIFO 算法的缓存命中率，特别是在缓存容量有限时，而其对 LFU 算法的改进效果随着缓存容量的增加而提升.

因此，当存在多个用户同时访问缓存时，Graph4Cache 也能够有效地处理多个缓存访问请求，提高传统缓存替换策略的命中率；也能有效提升传统的缓存替换算法在缓存容量较小时的性能. 因此，在涉及多个用户同时访问缓存的场景中，Graph4Cache 能够有效地处理多个缓存访问请求，并提高传统缓存替换策略的命中率，特别是能够显著改进传统算法在缓存容量有限时的性能. 通过序列推荐实验，我们验证了 Graph4Cache 能够有效地从访问序列中学习用户意图，并准确预测下一个数据访问对象. 此外，通过缓存模拟实验，我们验证了该模型能有效提升传统缓存替换算法在冷启动场景和多用户并发场景下的缓存命中率. 这 2 个实验共同证明了 Graph4Cache 的有效性.

5 总 结

本文研究通过提出 Graph4Cache 方法以解决现有缓存方案在缓存场景下无法获取用户画像以及无法捕获数据序列之间的高阶关系的问题. 首先，将用户的单个数据访问序列建模为有向图，从而学习序列内各数据项之间的转换关系. 其次，通过在有向图中引入虚拟节点来实现对序列的整体特征的学习. 最后，基于学习到的虚拟节点构建相关序列之间的跨序列图，从而结合多个用户的访问序列来学习数据项之间的高阶转换特征. 同时，由于跨序列图可以在有向图的学习过程中被构建，因此 Graph4Cache 是一种简单高效的缓存方案. 本文的实验结果表明，与其他多种图神经网络模型相比，Graph4Cache 能够更有效地学习用户的意图. 此外，Graph4Cache 在与经典缓存更新算法结合后，相较原有算法能够进一步提升缓存命中率.

作者贡献声明：尚晶提出了算法思路和实验方案并修改论文；武智晖、肖智文和张逸飞负责完成实验并撰写论文.

参 考 文 献

[1] Wang Yuqing, Yang Qiusong, Li Mingshu. A cache replacement policy based on instruction flow access pattern prediction[J]. Journal of Computer Research and Development, 2022, 59(1): 31−46 (in Chinese)
(王玉庆, 杨秋松, 李明树. 基于指令流访存模式预测的缓存替换策略[J]. 计算机研究与发展, 2022, 59(1): 31−46)

[2] Lekharu A, Jain M, Sur A, et al. Deep learning model for content aware caching at MEC servers[J]. IEEE Transactions on Network and Service Management, 2022, 19(2): 1413−25

[3] Fan Qilin, Li Xiuhua, Li Jian, et al. PA-Cache: Evolving learning-based popularity-aware content caching in edge networks[J]. IEEE Transactions on Network and Service Management, 2021, 18(2): 1746−57

[4] Jiang Weiwei, Luo Jiayun. Graph neural network for traffic forecasting: A survey[J]. Expert Systems with Applications, 2022, 207: 117921.

[5] Yu Junliang, Yin Hongzhi, Li Jundong, et al. Enhancing social recommendation with adversarial graph convolutional networks[J]. IEEE Transactions on Knowledge and Data Engineering, 2022, 34(8): 3727−39.

[6] Wang Wen, Zhang Wei, Liu Shukai, et al. Incorporating link prediction into multi-relational item graph modeling for session-based recommendation[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(3): 2683−96

[7] Gao Chen, Zheng Yu, Li Nian, et al. A survey of graph neural networks for recommender systems: Challenges, methods, and directions[J]. ACM Transactions on Recommendation System, 2023, 1(1): 1−51

[8] Hidasi B, Karatzoglou A, Baltrunas L, et al. Session-based recommendations with recurrent neural networks[J]. arXiv preprint, arXiv: 1511.06939, 2015

[9] Wu Shiwen, Sun Fei, Zhang Wentao, et al. Graph neural networks in recommender systems: A survey[J]. ACM Computing Surveys, 2022, 55(5): 1−37

[10] Wu Shu, Tang Yuyuan, Zhu Yanqiao, et al. Session-based recommendation with graph neural networks[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2019, 33((1):): 346−353

[11] Pan Zhiqiang, Cai Fei, Chen Wanyu, et al. Star graph neural networks for session-based recommendation[C]//Proc of the 29th ACM Int Conf on Information & Knowledge Management. Ireland: Association for Computing Machinery. 2020[2024-03-15]. doi: 10.1145/3340531.3412014

[12] Wang Ziyang, Wei Wei, Cong Gao, et al. Global context enhanced graph neural networks for session-based recommendation[C]//Proc of the 43rd Int ACM SIGIR Conf on Research and Development in Information Retrieval. China: Association for Computing Machinery. 2020[2024-03-15]. doi:10.1145/3397271.3401142

[13] Xia X, Yin H, Yu J, et al. Self-supervised hypergraph convolutional networks for session-based recommendation[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2021, 35(5): 4503−4511

[14] Lee B, An M, Lee S W. LRU-C: Parallelizing database I/Os for flash SSDs[J]. Proceedings of VLDB Endow, 2023, 16(9): 2364−76

[15] Chen Hao, Ruan Chaoyi, Li Cheng, et al. SpanDB: A fast, cost-effective LSM-tree based KV store on hybrid storage[C/OL]//Proc of the 19th USENIX Conf on File and Storage Technologies (FAST 21). 2021[2024-03-15]. www. usenix. org/conference/fast21/presentation/chen-hao

[16] Zhang Mi, Wang Qiuping, Shen Zhirong, et al. POCache: Toward robust and configurable straggler tolerance with parity-only caching[J]. Journal of Parallel and Distributed Computing, 2022, 167(C): 157−72

[17] Liu Zaoxing, Bai Zhihao, Liu Zhenming, et al. DistCache: Provable load balancing for large-scale storage systems with distributed caching[C/OL]//Proc of the 17th USENIX Conf on File and Storage Technologies (FAST 19). 2019[2024-03-15]. www. usenix. org/conference/fast19/presentation/liu

[18] Yang Zhong, Liu Yuanwei, Chen Yue, et al. Deep learning for latent events forecasting in content caching networks[J]. IEEE Transactions on Wireless Communications, 2022, 21(1): 413−28

[19] Rahman G M S, Peng M, Yan S, et al. Learning based joint cache and power allocation in fog radio access networks[J]. IEEE Transactions on Vehicular Technology, 2020, 69(4): 4401−4411

[20] Chen Yanhui, Huang Ling, Wang Changdong, et al. Hybrid-order gated graph neural network for session-based recommendation[J]. IEEE Transactions on Industrial Informatics, 2022, 18(3): 1458−1467

[21] Zangerle E, Pichl M, Gassler W, et al. #nowplaying music dataset: extracting listening behavior from Twitter[C]//Proc of the 1st Int Workshop on Internet-Scale Multimedia Management. 2014[2024-03-15]. doi: 2661714.2661719

[22] Liu Qiao, Zeng Yifu, Mokhosi R, et al. STAMP: Short-term attention/memory priority model for session-based recommendation[C]//Proc of the 24th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. 2018[2024-03-15]. doi: 3219819.3219950

[23] Ding Chengxin, Zhao Zhongying, Li Chao, et al. Session-based recommendation with hypergraph convolutional networks and sequential information embeddings[J]. Expert Systems with Applications, 2023, 223: 119875

[24] Sarwar B, Karypis G, Konstan J, et al. Item-based collaborative filtering recommendation algorithms[C]//Proc of the 10th Int Conf on World Wide Web. 2001[2024-03-15]. doi: 10.1145/371920.372071

[25] Rendle S, Freudenthaler C, Schmidt-Thieme L. Factorizing personalized Markov chains for next-basket recommendation[C]//Proc of the 19th Int Conf on World Wide Web. New York: ACM, 2010: 811−820

[26] Li Jing, Ren Pengjie, Chen Zhumin, et al. Neural attentive session-based recommendation[C]//Proc of 2017 ACM on Conf on Information and Knowledge Management. [2024-03-15]. doi: 10.1145/1772690.1772773

[27] Wang Meirui, Ren Pengjie, Mei Lei, et al. A Collaborative session-based recommendation approach with parallel memory modules[C]//

Proc of the 42nd Int ACM SIGIR Conf on Research and Development in Information Retrieval. New York: ACM, 2019: 345–354

[28] Xia Xin, Yin Hongzhi, Yu Junliang, et al. Self-supervised graph co-training for session-based recommendation[C]//Proc of the 30th ACM Int Conf on Information & Knowledge Management. New York: ACM, 2021: 2180–2190

[29] Chen Yao, Xiong Qi, Guo Ying. Session-based recommendation: Learning multi-dimension interests via a multi-head attention graph neural network[J]. Applied Soft Computing, 2022[2024-03-15]. doi: 10.1016/j.asoc.2022.109744

[30] Sheng Jinfang, Zhu Jiafu, Wang Bin, et al. Global and session item graph neural network for session-based recommendation[J]. Applied Intelligence, 2023, 53(10): 11737−11749



Shang Jing, born in 1978. PhD. Member of CCF. Her main research interests include big data, cloud computing, and databases.

尚　晶, 1978 年生. 博士. CCF 会员. 主要研究方向为大数据、云计算、数据库.



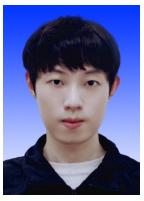
Wu Zhihui, born in 1978. Master. His main research interests include big data platforms, data processing, and databases.

武智晖, 1978 年生. 硕士. 主要研究方向为大数据平台、数据处理、数据库.



Xiao Zhiwen, born in 1993. PhD. Member of CCF. His main research interests include big data, machine learning, and cloud computing.

肖智文, 1993 年生. 博士. CCF 会员. 主要研究方向为大数据、机器学习、云计算.



Zhang Yifei, born in 1996. PhD. His main research interests include big data, machine learning, and cloud computing.

张逸飞, 1996 年生. 博士. 主要研究方向为大数据、机器学习、云计算.