**CCFSys2025 图计算系统设计大赛**

**设计方案**

|  |  |
| --- | --- |
| **项目名称：** | **一种以重叠为中心的增量** |
| **式动态超图匹配方法** |
| **及装置（OIDHMiner）** |

|  |  |
| --- | --- |
| **队伍名称：** | **图海拾贝** |
| **指导老师：** | **毛伏兵 秦磊华** |
| **所属学校：** | **华中科技大学** |
| **完成时间：** | **2025年9月1日** |

|  |
| --- |
| **一、设计方案概述** |
| 本设计方案提出了一种以重叠为中心的增量式动态超图匹配方法及装置，命名为OIDHMiner，旨在高效地处理动态超图模式匹配。超图作为图的泛化形式，能够自然地建模多方关系。在现实世界的应用中，数据常以超图结构表示，并随时间动态变化，使得能够更好的处理复杂系统的问题。然而，现有技术中的超图模式匹配方法大多依赖于静态、批量处理的范式，当超图发生频繁更新（如超边插入、删除和修改）时，这些方法因需要从头重新计算而导致效率低下。OIDHMiner系统通过引入一种以“重叠为中心”的增量式执行模型，有效地解决了这一技术难题。该系统由三个核心组件构成：动态化的无冗余编译器、并行执行与增量剪枝引擎，以及动态化的度感知数据存储。动态化的无冗余编译器负责生成超图的重叠交集图（OIG），并且根据其OIG生成模式重叠序列和执行计划，同时能够及时对OIG、模式重叠序列以及执行计划进行调整以适应超图的动态变化。并行执行与增量剪枝引擎负责根据执行计划，并行地进行模式匹配，并利用存储的部分匹配结果，对新的超边进行高效的增量检查和剪枝。动态化的度感知数据存储是一种基于散列表的数据结构，能够优化超图度值相关信息的存储和访问，以加速动态更新操作。当超图结构发生变化时，OIDHMiner能够对这些组件进行局部调整和更新，从而避免了大规模的重复计算，极大地提升了处理效率和系统实时性，特别适用于社交网络分析、生物信息学和网络安全等领域。 |
| **二、技术领域** |
| 本设计方案涉及技术属于计算机科学领域，特别是数据结构、算法和高性能计算的技术范畴。具体而言，本设计方案涉及超图（Hypergraph）数据模型下的模式匹配与挖掘技术，以及如何在一个持续变化的动态数据环境中高效地执行这些任务。该技术在分析复杂的多边关系数据时具有重要价值，其应用场景广泛，包括但不限于社交网络中的欺诈检测与社区发现、生物信息学中的基因和蛋白质相互作用分析、网络安全中的异常行为识别以及交通流量预测。 |
| **三、国、内外研究现状和发展动态** |
| 超图作为图的泛化形式，能够自然地建模多方关系，而不仅仅是传统的二元关系。这种能力使其在处理复杂系统时具有显著优势，例如，化学反应、人际交流或生物网络中的相互作用往往涉及多个实体。而在这些领域中，模式匹配通常被定义为子超图同构问题，其本身是一个NP-完全问题。  尽管现有技术中存在多种解决方案，但它们大多未能有效应对动态环境带来的挑战。传统的批量处理算法，例如基于子图同构的方法，当数据图（或超图）发生小规模更新时，需要从头开始重新计算所有匹配项。这种全量重算的过程成本高昂且不具有可扩展性，尤其对于那些规模巨大且频繁更新的图而言，其计算代价更为庞大。一些研究试图为图数据开发增量式算法，但它们通常只针对特定的图模式（如DAG）或特定类型的更新（如单边插入/删除）才能实现最优性能，并且其技术路线并未考虑超图特有的多边重叠关系。  此外，其他被称为“动态超图”的方法虽然存在，但其目的与超图模式匹配有所不同。例如，一些研究专注于动态超图结构学习（Dynamic Hypergraph Structure Learning），旨在根据数据特征或标签动态优化超图结构本身，而不是寻找预定义的模式实例。这些方法虽然也处理动态变化，但其核心任务是优化超图表示以支持学习任务，而非模式匹配。这种技术路线上的根本差异导致现有技术中存在一个显著的空白：缺乏一个专为处理动态超图上的模式匹配而设计的、高效且通用的增量式系统。  传统的模式匹配算法通常采用逐顶点匹配（match-by-vertex）或者逐超边匹配的框架，即递归地将查询超图的顶点映射到数据超图的顶点或者超边。这种方法将超边仅作为一种验证条件，而非计算的核心单元，导致搜索空间庞大且计算成本高，范式上的不足使得它难以在动态环境中进行高效的增量更新。  本设计方案OIDHMiner正是为了填补这一空白，通过引入一种全新的计算范式，从根本上优化了动态模式匹配的性能。 |
| **四、方案设计思路阐述** |
| 针对现有技术的不足，本方案提出了一种以重叠为中心的增量式动态超图匹配方法及装置OIDHMiner，其中包含三个关键系统单元（即动态化的无冗余编译器、并行执行和增量剪枝引擎和动态化更新的度感知数据存储结构（DDAL））。  该系统利用了一种新颖的以重叠为中心的增量式模型，通过充分利用超图模式挖掘具有的显著重叠相似性（即超边重叠中的多个顶点共同享有相同的关联超边）以及模型的增量特性来有效处理动态超图的模式匹配问题。具体而言，当我们需要进行动态超图模式匹配的时候，超图会经常随着时间的变化而做出增删改等操作，当需要改变模式时，动态化的无冗余编译器能够随时根据输入的模式生成匹配样式，然后应用到超图匹配中。并行执行和增量剪枝引擎能够根据动态化的无冗余编译器生成的以重叠为中心的执行计划将执行计划中的候选OIG节点与模式OIG中的节点进行验证匹配，在此过程中，并行执行和增量剪枝引擎还能够根据新增OIG节点的特性是否与模式OIG中的节点特性一致进行增量式剪枝操作，从而能够减少不必要的计算与验证开销。动态化更新的度感知数据存储结构（DDAL）能够根据超图的更新在原来的存储结构的基础上进行动态实时更新，避免了存储结构的重建，极大地减少了计算开销。   1. OIDHMiner系统主要包含三个关键系统单元（即动态化的无冗余编译器、并行执行和增量剪枝引擎和动态化更新的度感知数据存储（DDAL）)，如图1所示。     图1 OIDHMiner的系统架构  当OIDHMiner系统刚开始工作时，系统接受用户输入的指定模式超图以及初始的动态超图作为输入，并且将初始的动态超图存入动态化的度感知数据存储结构中，用于描述超图中超边之间的连接关系。其根据每条超边的度数进行分组，对于每条超边维护一个与其相连的超边列表，列表中根据超边的度数进行分组，便于高效匹配重叠顺序中的第一个顶点以及高效的剪枝操作。接着将用户指定的模式超图输入动态化的无冗余编译器中进行分析，构建重叠交集图（OIG），然后在此基础上生成重叠执行的顺序以及以重叠为中心的执行计划，在此过程中还使用基于组的剪枝技术来减少空重叠的冗余计算。最后，并行执行和增量剪枝引擎将根据生成的以重叠为中心的执行计划识别超图中的所有嵌入并且存入结果集中进行保存。  而当超图发生变化时，首先根据超图的度数以及关联关系变化修改动态化的度感知数据存储结构中超边之间的关系，然后后，并行执行和增量剪枝引擎将会根据新的以重叠为中心的执行计划，并且采用增量式剪枝方式识别更新后超图中的所有嵌入以及对结果集做出相应的更改。   1. OIDHMiner提出了动态化的无冗余编译器，用于高效的解决动态超图的结构时刻变化对OIG构建以及执行计划的问题。当需要匹配的模式发生变化时，可以在匹配任务前重新修改当前模式的结构，而不用重新加载整个任务。如图2所示。     图2 动态化无冗余编译器结构  （3）OIDHMiner提供了一种支持增量处理的并行执行引擎，其结构如图3所示。该引擎维护一个动态的部分匹配状态存储，只考虑了图中被修改部分和周围的子图。当超图发生变化时，引擎接收来自编译器的执行计划，并基于这些信息对部分匹配状态进行增量扩展和剪枝。其核心思想是：  1) 状态持久化：在超图变化期间保持有效的部分匹配，从而避免重复计算；  2) 增量验证：仅对受变化影响的局部匹配进行重新验证或扩展；  3) 协同剪枝：与动态编译器紧密协同，利用OIG信息进行早期剪枝，极大减少无效的计算路径。  4) 一个并行计算模块，用于将不同的初始匹配或受不同更新事件影响的分支分配至多个计算单元，以并行方式执行上述增量处理过程。    图3 并行执行与增量剪枝引擎技术结构  （4）OIDHMiner还提供了一种动态化的度感知数据存储结构。其核心在于维护一个以超边度数为键的倒排索引，并确保该索引能够随着超边插入、删除以及顶点增删（引起超边度数变化）而进行高效、局部的更新。该存储结构通过按度分组和维护分组邻接列表，使得执行引擎在候选生成时能够快速跳过大量不满足度数约束的超边，极大缩小搜索空间，为动态超图模式挖掘提供高效的数据供给。其结构图如图4所示。    图4 动态化度感知数据存储功能示意图 |
| 1. **具体实施方法** |
| 1. OIG的构建以及更新   为了便于说明超边不同操作对于OIG产生的影响以及详细的OIG修改实施步骤，我们给出初始超图的超边重叠、包含顶点情况等作为举例，如图5所示：    图5 初始超图举例  可以看到初始超图中包含四条超边（e1,e2,e3,e4），其中超边e1包含顶点u1,u2,u5,u6,超边e2包含顶点u1,u2,u3,u4,超边e3包含顶点u3,u4,u5,u6,u7,u8,超边e4包含顶点u7,u8,u9,u10  首先将每个超边作为一个OIG节点放置于OIG的顶层（即存入层顶点LV结构中），然后我们探索每一对超边节点之间的关系，若存在重叠部分，则将其重叠部分的顶点添加为下一层的一个节点，并且将这一对节点的箭头指向被添加的新节点。接着，我们继续合并下一层中的相同顶点，以避免在剩余层中它们之间进行冗余的交集计算，然后将下一层的顶点存储到LV中，继续执行上述步骤直至LV为空。最后根据上述步骤得到图5超图的OIG，如图6所示：    图6 举例初始超图的OIG  在构建过程中，我们需要排除空重叠的影响，因此采用基于组的OIG剪枝方式，利用当前层顶点的断开关系来剪枝剩余层顶点的断开关系。  例如，在图5中，e4与e3相连，而不与e1,e2相连。然后我们将同一层中的节点分成若干组，其中每组的前驱节点都相互连接，这样，我们只需确定组内的断开关系，由于各组之间的断开关系已由其前导组确定，例如，我们考虑第2层中的两个组（组2和组3），组2{ O5,O6,O7}的前驱节点O1,O2,O3超边节点之间两两重叠，因而划分为同一个组中，而由于e2和e4的重叠部分为空，因此O7和O8的重叠部分为空而没有划分为同一组。  A.添加超边  S1：添加顶层超边节点，生成后续重叠节点。  首先将新增的超边节点添加到OIG第一层节点中，然后将新增超边节点与原有超边节点之间重叠形成的结点全部添加在第二层节点，但是此时这些节点仍处于未分组状态。  S2：将第二层新增重叠节点添加到相应分组中，并且向下延伸OIG节点。  遍历所有第二层新增重叠节点，记录其除了新增超边节点以外的所有前驱超边节点，若这些节点与第二层某个分组中的所有前驱节点完全吻合，则将新增重叠节点全部加入该分组中，否则将新增重叠节点中处于前驱节点两两相连极大子集的部分划分为一个新的分组，从而将所有新增重叠节点划分为几个新的分组，从而能够保持每组的前驱节点都相互连接的分组特性。接着，对于新增重叠节点存在的分组，向下一层中添加新增重叠节点与其他节点的重叠节点，并且将相同的结点合并为一个节点，直至该组中只剩下一个节点而无法继续向下一层中添加节点。  例如，图7中的新增超边为e5，包含顶点u7,u8,u11,u12,仅与e3,e4存在重叠部分e7,e8，因此，其新增重叠节点除了新增超边节点以外的所有前驱超边节点为O3,O4，恰好与G3分组中的前驱节点一致，则将新增重叠节点全部加入该分组中。接着，向第三层中添加G3中每对节点的重叠节点，发现三个节点可以合并为一个节点，此时已经满足终止添加条件，因此完成OIG中节点的添加。  如果不按照这种增量式的方式对OIG进行修改，那么将会增加大量重复的计算开销。  原因：动态超图随着时间而时刻发生变化（超边的增加、删除以及修改等），那么将会导致相对应OIG也在随着时间而时刻发生变化，若每一次超图发生改变时，都采用OIG构建算法重新构建更新后超图的OIG，而我们又能够观察到更新后的超图实际上只是在原来的基础上进行了部分改变（即更新后的OIG仅在原来OIG基础上做出部分改变），那么将会导致未改变部分重复的OIG计算开销，若在超图规模较大的情况，这种重复的构建开销将会更加庞大。而如果采用增量式的修改方式，能够极大地减少重复的计算成本从而提高整体的计算效率。    图7 初始超图添加超边举例  S3：给更新后的OIG结点重新确定节点ID。  更新后的OIG中，新增节点没有唯一的标识——节点ID，且为了保持OIG图中节点ID的有序性，按照广度优先遍历顺序为每个OIG节点重新标识节点ID，最终更新后的OIG如图8所示：    图8 初始超图添加超边OIG  B.删除超边  S4：删除所有被删除超边节点的相关节点。  在第一层顶点中找到即将被删除的超边顶点，将其添加进入队列Q中（用于保存即将被删除的OIG结点）。接着进入循环，将Q队首节点弹出，将子节点加入队列Q中，然后将队首节点从OIG中删除，重复上述步骤，直至队列为空，则完成所有相关节点的删除。  例如，在图5的基础上删除e4，如图9所示：    图9 初始超图删除超边举例  在第一层节点中找到将要被删除的顶点O4，将其添加入队列Q中。接着，进入循环阶段：弹出队首元素O4，加入其子节点O8，删除O4；接着弹出队首元素O8，其子节点为空，删除O8；队列为空，则终止删除操作，完成相关节点的删除。  S5：给更新后的OIG结点重新确定节点ID。  更新后的OIG中，新增节点没有唯一的标识——节点ID，且为了保持OIG图中节点ID的有序性，按照广度优先遍历顺序为每个OIG节点重新标识节点ID，最终更新后的OIG如图10所示：    图10 初始超图删除超边OIG  C.修改超边  S6:将添加超边和删除超边视为原子操作，则修改超边为原子操作的组合序列（即删除被修改超边后重新进行添加）。其具体实施步骤可以参照上述添加超边和删除超边的步骤进行。   1. 以重叠为中心的执行计划生成   S7：生成基于拓扑排序的模式匹配顺序。  根据更新后OIG中节点的拓扑顺序，生成模式的匹配顺序。  以图10为例，在扩展O1时，我们只能匹配O1。而在扩展O2时，由于其前驱节点已被匹配，我们可以匹配O4。同样，在扩展O3时，由于其前驱节点已被匹配，我们可以匹配O5和O6，因此，示例OIG的重叠顺序为{O1, O2，O4,O3, O5，O6}。  S8：生成以重叠为中心的执行计划。  将更新后的模式OIG以及模式重叠顺序进行组合，以重叠顺序组织的集合操作和比较操作序列，用于计算重叠图中每个顶点的候选集，即为以重叠为中心的执行计划。在OIG中，超边顶点和重叠顶点的执行计划分别指导超图匹配过程中的候选生成和候选验证。   1. 并行执行引擎   S9：状态初始化：在初始超图H(t0)上，引擎使用静态OHMiner的算法完成一次完整的模式挖掘，并将所有找到的部分匹配和完整匹配存入状态存储库中。  S10：处理超边插入事件 (ΔE = +e):  A.候选匹配检索  引擎接收新超边e,它首先查询DAL，快速找到e的所有邻居超边（根据度数预分组）。然后，它在部分匹配状态存储中查找所有包含e的邻居超边的部分匹配实例,这些实例是可能被e扩展的候选。  B.增量扩展与验证  对于每一个候选部分匹配实例m'，引擎尝试将e加入其中，形成新的部分匹配m'' = m' ∪ {e}。  C.重叠剪枝（核心）  引擎并非重新计算m''的所有重叠，而是增量地计算新超边e与m'中已有超边之间新产生的重叠。它立即将这些新重叠的基数（size）与编译器提供的、更新后的模式OIG中对应位置的要求进行比对。如果任一重叠不满足条件（例如，|e ∩ e\_i| ≠ |pe\_j ∩ pe\_k|），则整个m''被立即剪枝。否则，m''被验证为新的有效部分匹配，加入状态存储库。  S11:处理超边删除事件 (ΔE = -e):  A.状态失效  引擎接收被删除的超边e,它直接在部分匹配状态存储中执行一个查询，找出所有包含超边e的部分匹配实例。  B.批量移除  将这些实例全部标记为无效并从存储中移除。这样操作简单高效，复杂度与包含e的匹配实例数量成正比。  S12:并行执行：  引擎采用任务级并行。每个初始候选超边、或每个由更新事件触发的增量扩展分支都可以被封装成一个独立的任务。这些任务被放入一个共享队列中，由一组工作线程通过动态调度（如工作窃取）并行执行，充分利用多核资源处理连续的数据流。并行性隐藏了单个增量验证任务的延迟，极大地提高了系统在动态环境下的吞吐量和响应速度。   1. 动态化的度感知数据存储   动态化的度感知存储结构通过按度分组和维护分组邻接列表，使得执行引擎在候选生成时能够快速跳过大量不满足度数约束的超边，极大缩小搜索空间，为动态超图模式挖掘提供高效的数据供给。例如，对图11的超图可以计算存储数据如表1所示：    图11 计算度数和展示存储的超图   |  |  |  | | --- | --- | --- | | Hyperedge | Degree | DAL | | e1 | 6 | {{e2,e4},{e3,e5}} | | e2 | 6 | {{e1},{e3,e5}} | | e4 | 6 | {{e1},{e5}} | | e3 | 8 | {{e1,e2,e4},{e5}} | | e5 | 8 | {{e1,e5},{e3}} |   表1 图11实例超图的DAL  S13:初始化过程：  系统加载初始超图静态快照，为每个超边e计算其度数d(e)。然后，构建主哈希表DAL\_Map: { degree -> list of hyperedges }。同时，为每个超边e构建其分组邻接列表：将其所有邻居超边adj(e)按它们的度数进行分组，得到形如{ degree\_i -> [neighbors with degree\_i] }的结构。  S14：处理超边插入事件 (ΔE = +e):  A.计算度数  计算新超边e的度数d(e)。  B.更新主结构  将e添加到DAL\_Map[d(e)]对应的列表中。  C.更新邻接信息  遍历e中包含的所有顶点，对于每个顶点v，遍历v的所有关联超边（即e的新邻居）。对于每个邻居超边e\_neighbor，将其加入到e的分组邻接列表中d(e\_neighbor)对应的分组。同时，将e加入到e\_neighbor的分组邻接列表中d(e)对应的分组。这些更新操作是局部的，仅影响新超边自身和其邻居超边，效率极高。  举例说明来看，新超边e6（度数为4）加入，它包含顶点vx。vx原本关联着超边e1（度数为6）和e2（度数为8）。更新后：DAL\_Map[4]包含e6；e6的邻接列表新增分组6: [e1]和8: [e2]；同时，e1的邻接列表在分组4:中加入e6，e2的邻接列表在分组4:中也加入e6。邻接列表更新过程如图12所示：    图12 邻接链表更新过程  更新前后DAL\_MAP对比图如下图图13所示：    图13 更新前后DAL\_MAP对比图  S15:处理超边删除事件 (ΔE = -e):  A.更新主结构  从DAL\_Map[d(e)]对应的列表中移除e。  B.更新邻接信息  遍历e的所有邻居超边（从其分组邻接列表中可获知），对于每个邻居超边e\_neighbor，从其分组邻接列表中d(e)对应的分组里移除e。  C.清理  清理并释放e自身的存储空间。  S16：处理顶点事件（引起超边度数变化） (ΔV):  这是最复杂的情况，插入或删除一个顶点v会影响所有包含v的超边E\_v的度数。  A.标记  标记所有e ∈ E\_v，它们的度数d(e)将变为d'(e)。  B.原子性更新  对于每一条受影响的超边e，执行：  a) 从DAL\_Map[d(e)]中移除e。  b) 将e加入到DAL\_Map[d'(e)]中。  c) 通知所有e的邻居超边：邻居需要将e从它们邻接列表中原先d(e)的分组中移除，并加入到新的d'(e)的分组中。  d) 更新e自身的度数标签。  通过度数分组的变化，自动维护了候选生成所需的过滤信息。但与此同时一次顶点更新可能引发大量超边的度数变化及其邻接信息的连锁更新（写放大）。需要精心设计批量处理策略来缓解性能冲击。  S17：查询接口：  为执行引擎提供高效查询接口，例如：get\_hyperedges\_by\_degree(k)：直接返回DAL\_Map[k]。get\_neighbors\_with\_degree(e, k)：直接访问e的分组邻接列表，返回k分组下的所有邻居超边，无需遍历整个邻接表并进行度数过滤，这是相比传统存储结构的巨大优势。  在代码实现中，并不是按照理论分析的用hypergraph上oig和编译器上的oig进行匹配的方式来执行，而是按照等价的方式实现，当添加 修改删除超边时，我们只考虑和这个超边相邻的超边及其超边对应的点所构成的子超图，在这个子超图上时用编译器生成的查询计划来匹配。因为对于超图中的其他部分，不会受图动态变化的影响，在这些部分上的匹配结果不会有变化。只有超边相邻的子超图上会发生改变，当加新边时，子超图可能出现新的子图，当删除边时，子超图中的模式会被删除，当修改超边时，相当于先删除子超边，再加入新的子超边，分别按照删除和插入的模式计算即可得到修改导致的变化。值的注意的是，可能出现在字图中统计的模式不包含被操作的超边的情况，所以我们在子超图上匹配到模式时，要通过判断匹配部分是否包含有被操作超边来判断匹配到的模式是否有效，如果有效才被用作计数。 |
| **六、设计方案附图说明** |
| 图1    图2    图3    图4    图5    图6    图7    图8    图9    图10    图11    图12    图13  图1是本设计方案所述OIDHMiner系统的整体架构示意图，展示了系统由动态化的无冗余编译器、并行执行与增量剪枝引擎、动态化度感知数据存储（DDAL）三大核心模块组成及其协同工作机制。  图2是动态化无冗余编译器的结构示意图，展示了其接收模式超图、构建与维护重叠交集图（OIG）、生成执行计划等核心功能流程。  图3是并行执行与增量剪枝引擎的结构示意图，展示了其状态存储、增量扩展、重叠剪枝及并行调度等关键技术组件。  图4是动态化度感知数据存储（DDAL）的功能示意图，展示其按度分组、两级索引结构及增量更新机制。  图5是一个初始超图的示例图，用于说明超边及其顶点组成关系，作为后续OIG构建与更新的基础实例。  图6是基于图5初始超图构建的重叠交集图（OIG）示意图，展示了超边及其重叠关系的层次化有向无环图结构。  图7是在图5初始超图中添加一条新超边e5后的超图示例，用于说明超边插入对OIG的影响。  图8是在图6基础上插入超边e5后更新得到的OIG示意图，展示了增量更新后的节点结构与分组情况。  图9是在图5初始超图中删除超边e4后的超图示例，用于说明超边删除对OIG的影响。  图10是在图6基础上删除超边e4后更新得到的OIG示意图，展示了删除操作后的OIG结构。  图11是一个用于说明度感知数据存储（DAL）构建过程的超图实例，展示了超边及其度数关系。  图12是超边插入时邻接链表更新过程的示意图，展示了DDAL在插入新超边时的局部更新机制。  图13是DDAL中主哈希表（DAL\_Map）在超边插入前后对比示意图，展示了按度分组结构的动态更新过程。 |