Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS-10)

Edge Partitioning in Parallel Structured Duplicate Detection

Rong Zhou[1] and **Tim Schmidt**[1] and **Eric A. Hansen**[2] and **Minh B. Do**[1] and **Serdar Uckun**[1]

[1]Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
rzhou, tschmidt, minhdo, uckun at parc.com

[2]Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
hansen at cse.msstate.edu

Introduction

Heuristic-search planners that use A* and related graph search algorithms must be parallelized to harness advances in computing power that are based on increasing use of multi-core processors. Although a graph can always be converted to an equivalent *tree* that can be easily searched in parallel, such a conversion increases the size of the search space *exponentially*, and the resulting overhead is hard to justify in the context of parallel search for which the speed-up ratio is bounded by the number of parallel processes, a *polynomial* resource in most practical settings. A more direct approach to parallelizing *graph* search is needed.

The challenge in parallelizing graph search is *duplicate detection*, which requires checking newly generated nodes against the set of already visited nodes. If performed naively, duplicate detection may require excessive synchronization among concurrent search processes (e.g., to maintain the open and closed lists of A^*). Here we show how *edge partitioning*, a technique that was developed originally for reducing the number of time-consuming disk I/O operations in external-memory search, can be used in a parallel setting to reduce the frequency with which search processes need to synchronize with one another, effectively reducing the primary source of overhead in parallel graph search.

Structured duplicate detection

Structured duplicate detection (SDD) (Zhou and Hansen 2004) was originally developed as an approach to external-memory graph search that leverages local structure in a graph to partition stored nodes between internal memory and disk in such a way that duplicate detection can be performed immediately, during node expansion, instead of being delayed. SDD uncovers local structure in a graph by using a state-space projection function that maps nodes in the original space into an abstract search space. Nodes are divided into "$n$blocks," where an $n$block corresponds to a set of nodes that map to the same abstract node. The concept of *duplicate-detection scope* allows SDD to check duplicates against a fraction of stored nodes, and still guarantee that all duplicates are found.

Parallel structured duplicate detection

Parallel structured duplicate detection (PSDD) (Zhou and Hansen 2007a) is an extension of SDD that leverages the same local structure to reduce the amount of synchronization needed in parallel search. PSDD assigns idle processors $n$blocks to expand whose duplicate detection scopes are pairwise disjoint, since they can be searched in parallel without any synchronization. Note that two duplicate-detection scopes are disjoint if they do not share any $n$blocks. Synchronization among processes is only necessary after a process has finished expanding all nodes in its current $n$block and needs to acquire a new $n$block. PSDD only needs a single lock, controlling the manipulation of the abstract graph, and the lock only needs to be acquired by a process when finding a new free $n$block to search.

Edge partitioning

Edge partitioning (Zhou and Hansen 2007b) is an enhancement of SDD that is guaranteed to localize memory references even in cases when a search graph has no apparent local structure. In edge partitioning, the duplicate-detection scope of an $n$block is no longer defined as the set of all successor $n$blocks in the abstract graph; instead, it is defined as the single successor $n$block along a particular outgoing edge in the abstract graph. At each stage, the operators corresponding to a different abstract edge are applied; all operators corresponding to the abstract edge are applied to every node in the $n$block before any other operators are considered. At the next stage, a different outgoing abstract edge is considered, and a different set of operators is applied to the same set of nodes. Eventually, all operators are applied to the $n$block of nodes and the nodes become fully expanded.

Parallel Edge Partitioning

Our new approach to parallel graph search, which we call *parallel edge partitioning* (PEP), is based on the intuition that reducing the size of duplicate-detection scopes has the effect of reducing the degree of contention among search processes, which increases the degree of concurrency that is possible in parallel search.

Note that if the state-space projection function used by PSDD results in a fully connected abstract graph, there would be a single duplicate-detection scope that corresponds to the entire graph, and PSDD would be unable to search in parallel because it would be impossible to find disjoint scopes. But with edge partitioning, any duplicate-detection scope contains only a *single* $n$block. Therefore, the number of concurrent search processes allowed in PEP is the number of nodes in the abstract graph, regardless of its structure. For any state-space projection function, it can be shown that PEP is optimal in the sense that no other algorithm can achieve a higher degree of concurrency without compromising duplicate detection.

PEP reduces the need for load balancing, because the same $n$block can be expanded by multiple processes in parallel, which is not the case for PSDD. This means that if one $n$block is significantly larger than others, PEP can assign multiple processes to work on it, one for each outgoing abstract edge. Intuitively, this is equivalent to dividing a large task into smaller pieces; the more pieces there are, the easier it is to balance load and keep all processes busy.

PEP is deadlock-free by design, with a simple proof: it breaks one of the four necessary conditions for a deadlock. Recall the four conditions are (1) mutual exclusion, (2) hold and wait, (3) no preemption, and (4) circular wait. The condition that is never satisfied in parallel edge partitioning is "hold and wait," because once a search process gets hold of a single $n$block (a unit resource in PEP) that is the duplicate-detection scope of an abstract edge, it has all that is needed to proceed with node expansions and there is no waiting period for a deadlock to occur.

Edge partitioning in domain-independent planning

In previous work, edge partitioning was evaluated as a technique for disk-based search, in which disk I/O operations are a major source of overhead. Those experiments (Zhou and Hansen 2007b) showed that, in most cases, edge partitioning is slower than SDD, because it needs more disk I/O for swapping in (out) nodes stored on disk (in RAM). When used for parallelization of purely internal-memory search, however, our new experiments show that edge partitioning can be significantly faster than PSDD, for two reasons.

First, by reducing any duplicate-detection scope to a single $n$block, edge partitioning enforces excellent memory-reference locality, which makes the code run faster on modern processors. Second, and more importantly for domain-independent planning, edge partitioning reduces the overhead for precondition checking that determines operator applicability when a node is expanded. For many planning problems, the set of grounded operators can become fairly large. Fortunately, edge partitioning performs *operator abstraction* on top of state-space abstraction by pre-computing the set of applicable operators for each $n$block, and if an operator fails to meet the precondition for an $n$block, edge partitioning can safely ignore it without any testing during node expansions.

Computational Results

We implemented parallel edge partitioning in a STRIPS planner that uses as its underlying search algorithm *breadth-first heuristic search* (Zhou and Hansen 2006),which is more memory-efficient than A*. Our planner uses forward state-space planning to find optimal sequential plans, guided by an admissible pattern database heuristic. We tested eight benchmark domains from the biennial planning competition. On average, PEP needs 10 times more incremental node expansions than PSDD needs full node expansions, yet PEP is still 27% faster with 7 threads. On average, the size of the abstract graph used in PEP is 10 times smaller than that used in PSDD. This shows PEP can be more effective than PSDD, even when using a coarser abstraction function.

We also compared PEP with recent parallel search algorithms, including HDA* (Kishimoto, Fukunaga, and Botea 2009) and PBNF (Burns et al. 2009). While using more memory, both ran slower than PEP on all thread counts. We also tested PEP on hard instances of the 15-Puzzle formulated as STRIPS planning problems. Previously, the best domain-independent solver (Haslum et al. 2007) can solve 93 of them in hours. PEP can solve 95 of them in minutes, using a much weaker admissible heuristic. None of the hard instances can be solved by HDA* or PBNF, since both ran out of memory well before any instance could be solved.

Conclusion and Future Work

We have described a parallel version of the edge partitioning technique in structured duplicate detection that is guaranteed to be effective even when the underlying search graph has no local structure. In domain-independent planning, it also provides significant speedup from faster precondition checking. As a result, PEP runs faster than PSDD even on a single thread. In the future, we will integrate this technique with external-memory graph search, which can benefit from both I/O parallelism and parallelism in internal-memory search, allowing larger graph-search problems to be solved.

References

Burns, E.; Lemons, S.; Zhou, R.; and Ruml, W. 2009. Best-first heuristic search for multi-core machines. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, 449–455.

Haslum, P.; Helmert, M.; Bonet, B.; Botea, A.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*, 1007–1012.

Kishimoto, A.; Fukunaga, A.; and Botea, A. 2009. Scalable, parallel best-first search for optimal sequential planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*, 201–208.

Zhou, R., and Hansen, E. 2004. Structured duplicate detection in external-memory graph search. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, 683–688.

Zhou, R., and Hansen, E. 2006. Breadth-first heuristic search. *Artificial Intelligence* 170(4-5):385–408.

Zhou, R., and Hansen, E. 2007a. Parallel structured duplicate detection. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, 1217–1223.

Zhou, R., and Hansen, E. 2007b. Edge partitioning in external-memory graph search. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2410–2416.