Cost-Optimal External Planning

Stefan Edelkamp* **and Shahid Jabbar***
Computer Science Department,
Otto-Hahn Strasse 14, University of Dortmund,
Dortmund 44227, Germany
{stefan.edelkamp, shahid.jabbar}@cs.uni-dortmund.de

Abstract

This paper considers strategies for external memory based optimal planning. An external breadth-first search exploration algorithm is devised that is guaranteed to find the cost-optimal solution. We contribute a procedure for finding the upper bound on the locality of the search in planning graphs that dictates the number of layers that have to be kept to avoid re-openings.

We also discuss an external variant of Enforced Hill Climbing. Using relaxed-plan heuristic without helpful-action pruning we have been able to perform large explorations on metric planning problems, providing better plan lengths than have been reported earlier. A novel approach to plan reconstruction in external setting with linear I/O complexity is proposed. We provide external exploration results on some recently proposed planning domains.

Introduction

In recent years, AI Planning has seen significant growth in both theory and practice. Most of these approaches revolve around search. The underlying transition graph is searched for the state where desired goal criteria are fulfilled. *PDDL* (McDermott & others 1998), for Planning Domain Definition Language, provides a common framework to define planning domains and problems. Starting from a pure propositional framework, it has now grown into accommodating more complex planning problems. In *metric planning*, we see a numerical extension to the STRIPS planning formalism, where actions modify the value of numeric state variables. The task is then to

- find a path from an initial state to a state where all goal criteria are fulfilled
- *additionally* optimize an objective function

Unfortunately, as the planning problems get complicated, the size of the state and the number of states grows significantly too - easily reaching the limits of main memory capacity. Having a systematic mechanism to flush the already seen states to the disk can circumvent the problem. Algorithms that utilize secondary storage devices have seen significant success in single-agent search, e.g., in (Korf & Schultze 2005), we see a complete exploration of the state space of 15-puzzle made possible by utilizing 1.4 Terabytes of secondary storage. In (Jabbar & Edelkamp 2006), we see a successful application of external memory heuristic search for LTL model checking. An important issue that arises in the designing of external memory algorithms is the removal of duplicate nodes to avoid re-expansions. (Zhou & Hansen 2004) proposed structured duplicate detection where the state space structure is exploited to define a partition on the state space. Removal of duplicates is then restricted only to the neighboring partitions. The rest of the partitions can then be removed from the main memory and flushed to the disk. Among the reported results are applications of the approach on STRIPS planning problems. Unfortunately, the approach is limited only to the state spaces where the neighboring partitions and the active partition can fit into the main memory.

The paper is structured as follows. We first discuss metric planning problems with linear expressions. An overview of the external memory model is presented afterward. Then, we introduce external BFS algorithm for implicit undirected graphs. Directed graphs are treated in the next section where we contribute a formal basis to determine the locality of planning graphs which dictates the number of previous layers to look at during duplicate removal to avoid re-expansion of nodes. Introducing heuristic guidance in the form of relaxed plan computation, we discuss an external memory variant of enforced hill climbing for suboptimal metric planning problems. This treatment is followed by providing an algorithmic solution for the optimality problem. Finally, we contribute large exploration results to validate our approach and discuss future extensions.

Metric Planning

A *planning problem* can be regarded as a state space exploration problem in implicit graphs. A vertex or a state in this graph consists of a set of *facts* that holds in that state. Successors are generated by applying operators. An operator O is a 3-ary tuple $\langle pre(O), add(O), del(O)\rangle$, where $pre(O)$ is a set of pre-conditions for applying a; $add(O)$, the set of new facts to be added to the new state; and, $del(O)$ being the set of facts to be deleted from the new state. We can now for-

*Supported by the German Research Foundation (DFG) projects *Heuristic Search* Ed 74/3 and *Directed Model Checking* Ed 74/2.

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :precondition (and (at ?a ?c1)
    (>= (fuel ?a) (* (dist ?c1 ?c2) (slow-burn ?a))))
 :effect (and (not (at ?a ?c1)) (at ?a ?c2)
    (increase total-fuel-used
         (* (dist ?c1 ?c2) (slow-burn ?a)))
    (decrease (fuel ?a)
         (* (dist ?c1 ?c2) (slow-burn ?a))))
)
```

Figure 1: An metric operator in PDDL, Level 2.

mally define the planning task as $\mathcal{P} = (V, \mathcal{I}, \mathcal{G}, \mathcal{O})$, where V is the set of states, $\mathcal{I} \in V$, the initial state, $\mathcal{G} \subseteq V$ the set of goal states, and $\mathcal{O} : V \rightarrow V$, the set of operator.

Metric planning (Fox & Long 2003) involves reasoning about continuous state variables and arithmetic expressions. Preconditions are of the form *exp* $\otimes$ *exp*$'$, where *exp* and *exp*$'$ are arithmetic expressions over $\{+, -, *, /\}$ and $\otimes \in \{\geq, \leq, >, <, =\}$. Assignments are of the form $v \oplus$ *exp* with variable head v and assignment operator $\oplus$. An example for a metric operator in PDDL notation is given in Figure 1. Metric planning is a considerable extension in language expressiveness. As one can encode arbitrary Turing machine computations into real numbers even the decision problem becomes undecidable (Helmert 2002). For this text, we restrict preconditions and effects to be linear expressions. This is not a severe limitation in planning practice, as all planning benchmarks released so far can be compiled to a linear representation (Hoffmann 2003).

Optimization in metric planning calls for improved exploration algorithms. Besides A*, depth-first branch-and-bound is a common choice. For efficient optimization in metric domains, both state-of-the-art planners LPG-TD (Gerevini, Saetti, & Serina 2006) and SGPlan (Wah & Chen 2006) apply Lagrangian optimization techniques to gradually improve a (probably invalid) first plan. Both planners extend the relaxed planning heuristic as proposed in Metric-FF.

A recently added feature of PDDL is the facility to define plan and preference constraints (Gerevini & Long 2005). Annotating individual constraints with preferences models soft constraints. For example, if we prefer block a to reside on the table after the plan's execution, we write `(preference p (on-table a))` with a validity check `(is-violated p)` in the plan objective. Such propositions are interpreted as natural numbers that can be included into the plan's *cost function*. This (linear) function allows planners to search for cost-optimal plans. For planning with preferences the cost function first scales and then accumulates the numerical interpretation of the propositions referring to the violation of the preference constraints. Even though we conduct experiments in problems with preference constraints by using the PDDL3-to-PDDL2 compilation approach of (Edelkamp 2006), the input of the exploration engine is Level 2 metric planning problem.

External Exploration

The standard model for comparing the performance of external algorithms consists of a single processor, a small internal memory that can hold up to M data items, and an unlimited secondary memory. The size of the input problem (in terms of the number of records) is abbreviated by N. Moreover, the *block size* B governs the bandwidth of memory transfers. It is usually assumed that at the beginning of the algorithm, the input data is stored in contiguous block on external memory, and the same must hold for the output. Only the number of block reads and writes are counted, computations in internal memory do not incur any cost. The single disk model for external algorithms is devised by (Aggarwal & Vitter 1988). It is convenient to express the complexity of external-memory algorithms using a number of frequently occurring primitive operations: *scan*(N) for scanning N data items, with an I/O complexity of $\Theta(N/B)$ and *sort*(N) for external sorting, with an I/O complexity of $\Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$. In our case, N is replaced with the number of nodes $|V|$ or the number of edges $|E|$.

Munagala and Ranade's algorithm (Munagala & Ranade 1999) for explicit Breadth-First Search has been adapted for implicit graphs. The new algorithm is known as *delayed duplicate detection* for *frontier search*. It assumes an undirected search graph. The algorithm maintains BFS layers on disk. Let *Open*(i) represents the set of states at layer i. Layer *Open*$(i-1)$ is scanned and the set of successors are put into a buffer of size close to the main memory capacity. If the buffer becomes full, internal sorting followed by a scanning generates a sorted duplicate-free state sequence in the buffer that is flushed to the disk. This results in a file with states belonging to depth i stored in the form of sorted buffers. To remove the duplicates, *external sorting* is applied to unify the buffers into one sorted file. Due to sorting, all duplicates will come close to each other and a simple scan is enough to generate a duplicate free file. One also has to eliminate/subtract previous layers from *Open*(i) to avoid re-expansions. In (Munagala & Ranade 1999), the authors argue that for undirected graphs, subtracting two previous layers is enough to guarantee that no state is expanded twice.

The corresponding pseudo-code is shown in Figure 2. *Succ* represents the successor generation function while A-sets correspond to temporary files. Note that the partition of the set of successors into blocks is implicit. Termination is not shown, but imposes no additional implementation problem. As with the algorithm of Munagala and Ranade, delayed duplicate detection applies $O(sort(|Succ(Open(i-1))|) + scan(|Open(i-1)| + |Open(i-2)|))$ I/Os. Since each edge contributes to one state, $\sum_i |Succ(Open(i))| = O(|E|)$ and $\sum_i |Open(i)| = O(|V|)$. This gives a total I/O complexity is $O(sort(|E|) + scan(|V|))$ I/Os.

To highlight the optimality of the approach we refer the reader to (Aggarwal & Vitter 1988), who showed a matching I/O lower bound for external sorting. (Arge, Knudsen, & Larsen 1993) extended this work to the issue of duplicate detection, which is a necessity when substituting the hash table while allowing the elimination of repeated states.

The sorting complexity can be improved in practice by us-

Procedure External-BFS
$Open(-1) \leftarrow \emptyset, Open(0) \leftarrow \{\mathcal{I}\}$
$i \leftarrow 1$
while $(Open(i-1) \neq \emptyset)$
$A(i) \leftarrow Succ(Open(i-1))$
$A'(i) \leftarrow$ *remove duplicates from* $A(i)$
$Open(i) \leftarrow A'(i) \setminus (Open(i-1) \cup Open(i-2))$
$i \leftarrow i+1$

Figure 2: External Breadth-First Search with delayed duplicate detection; $\mathcal{I}$ is the initial state and *Succ* is the successor generation function.

ing a Hash-based delayed duplicate detection scheme. Frontier Search with Hash-based delayed duplicate detection has been used to fully explore the 15-Puzzle with 1.4 Terabytes of harddisk in about three weeks (Korf & Schultze 2005). The algorithm is similar to the internal *frontier search* algorithm (Korf & Zhang 2000) that has been used for solving multiple sequence alignment problem.

Locality in Planning Domains

A crucial issue in external memory algorithms is the removal of duplicates. Since there is no hash table involved, duplicate nodes have to be removed by scanning previous layers. The number of layers sufficient for full duplicate detection depends on a property of the search graph called *locality*. Let $\mathcal{I}$ be the start state of the problem. For integer weighted problem graphs, the *locality* is defined as the maximum $\max\{\delta(\mathcal{I},u) - \delta(\mathcal{I},v), 0\}$ of all nodes u, v, with v being a successor of u and $\delta(u,v)$ being the shortest path distance between two nodes u and v. For undirected graphs we always have that $\delta(\mathcal{I},u)$ and $\delta(\mathcal{I},v)$ differ by at most one so that the locality is 1.

Let l be the graph's locality, and z the number of stored layers. In breadth-first search, when layer m is expanded, all previous layers with depth smaller than m have been closed, and are known by their optimal depth value. Thus, if a node u at level m is expanded, and its successor v has a shorter optimal distance to $\mathcal{I}$, i.e., $m = \delta(\mathcal{I},v) < \delta(\mathcal{I},u) = m'$, then v must have been encountered earlier in the search, in the worst case at layer $m' = m - l$. The re-expansion of v will be avoided *iff* it is contained in the stored layers $m - z \ldots m-1$; i.e., $z \geq l$. This is the basis of the following theorem due to (Zhou & Hansen 2006):

Theorem 1 *(Locality Determines Boundary) The number of previous layers of a breadth-first search graph that need to be retained to prevent duplicate search effort is equal to the locality of the search graph.*

As a special case, in undirected graphs, the locality is 1 and we need to store the immediate previous layer only to check for duplicates.

The condition $\max\{\delta(\mathcal{I},u) - \delta(\mathcal{I},v), 0\}$ over all nodes u, v, with v being a successor of u is not a graph property. So the question is, can we find a sufficient condition or upper bound for it? The following theorem proves the existence of such a bound.

Theorem 2 *(Upper-Bound on Locality) The locality of a uniformly weighted graph for breadth-first search can be bounded by the minimal distance to get back from a successor node v to u, maximized over all u. In other words, with $v \in \mathrm{Succ}(u)$, we have*

$$\max_{u \in V}\{\delta(v,u)\} \geq \max_{u \in V}\{\delta(\mathcal{I},u) - \delta(\mathcal{I},v), 0\}$$

Proof: For any nodes $\mathcal{I}, u, v$ in a graph, the triangular property of shortest path $\delta(\mathcal{I},u) \leq \delta(\mathcal{I},v) + \delta(v,u)$ is satisfied, in particular for $v \in Succ(u)$. Therefore $\delta(v,u) \geq \delta(\mathcal{I},u) - \delta(\mathcal{I},v)$ and $\max_{u \in V}\{\delta(v,u)\} \geq \max_{u \in V}\{\delta(\mathcal{I},u) - \delta(\mathcal{I},v)\}$. In positively weighted graphs, we have $\delta(v,u) \geq 0$ such that $\max_{u \in V}\{\delta(v,u)\}$ is larger than the locality.

As for graphs without self-loops with $v \in Succ(u)$, we have $\max_{u \in V}\{\delta(v,u)\} = \max_{u \in V}\{\delta(u,u)\} - 1$. Hence, in order to bound the locality, we have to look for the largest minimal cycle in the graph.

The question then arises is: How can we find out the largest minimal cycle in an implicitly given graph as they appear in action planning? The answer to this question lies in the rules or operators in a state space. Without loss of generality, we consider STRIPS planning operators in the form of $\langle pre(O) add(O), del(O)\rangle$, representing preconditions, add, and delete lists for an operator O. A duplicate node in an implicit graph appears when a sequence of operators applied to a state generates the same state again, i.e., they cancel the effects of each other. Hence the following definition:

Definition 1 *(*no-op *Sequence) A sequence of operators $O_1, O_2, \ldots, O_k$ is a* no-op sequence *if its application on a state produces no effects, i.e., $O_k \circ \ldots \circ O_2 \circ O_1 =$* no-op*,*

This definition provides us the basis to bound the locality of the implicit graphs in the following theorem. It generalizes undirected search spaces, in which for each operator O_1 we find an inverse operator O_2 such that $O_2 \circ O_1 =$ *no-op*.

Theorem 3 *(*no-op *Sequence determines Locality) Let $\mathcal{O}$ be the set of operators in the search space and $l = |\mathcal{O}|$. If for all operators O_1 we can provide a sequence $O_2, \ldots, O_k$ with $O_k \circ \ldots \circ O_2 \circ O_1 =$* no-op*, where* no-op *is the identity mapping, then the locality of the implicitly generated graph is at most $k-1$.*

Proof: If $O_k \circ \ldots \circ O_2 \circ O_1 =$ *no-op* we can reach each state u again in at most k steps. This implies that $\max_{u \in V}\{\delta(u,u)\} = k$. Theorem 2 shows that $\max_{u \in V}\{\delta(u,u)\} - 1$ is an upper bound on the locality.

The condition $O_k \circ \ldots \circ O_2 \circ O_1 =$ *no-op* can be tested in $O(l^k)$ time. It suffices to check that the cumulative add effects of the sequence is equal to the cumulative delete effects. Using the denotation by (Haslum & Jonsson 2000), the cumulative add C_A and delete C_D effects of a sequence can be defined inductively as,

$$\begin{aligned} C_A(O_k) &= A_k \\ C_D(O_k) &= D_k \quad \text{and,} \end{aligned}$$

Procedure External-EHC-BFS(u, h)
  Open$(-1, h) \leftarrow \emptyset$, *Open*$(0, h) \leftarrow u$, $i \leftarrow 1$
  while (*Open*$(i-1, h) \neq \emptyset$)
    $A(i) \leftarrow$ *Succ*(*Open*$(i-1, h)$)
    forall $v \in A(i)$
      $h' =$ *Heuristic*(v)
      if $h' < h$ **return** (v, h')
    $A'(i) \leftarrow$ *remove duplicates from* $A(i)$
    for *loc* $\leftarrow 1$ **to** *locality*
      $A'(i) \leftarrow A'(i) \setminus$ *Open*$(i -$ *loc*$)$
    Open$(i) \leftarrow A'(i)$
    $i \leftarrow i + 1$
  return ∞

Figure 3: External BFS for External Enforced Hill Climbing; u is the new start state with the heuristic estimate h.

Procedure External Enforced Hill-Climbing
  $u \leftarrow \mathcal{I}$, $h =$ *Heuristic*$(\mathcal{I})$
  while $(h \neq 0)$
    $(u', h') \leftarrow$ *External-EHC-BFS*(u, h)
    if $(h' = \infty)$ **return** $\emptyset$
    $u \leftarrow u'$
    $h \leftarrow h'$
  return *ConstructSolution*(u)

Figure 4: External Enforced Hill-Climbing; $\mathcal{I}$ is the start state.

$$\begin{aligned} C_A(O_1, \ldots, O_k) &= (C_A(O_1, \ldots, O_{k-1}) - D_k) \cup A_k \\ C_D(O_1, \ldots, O_k) &= (C_D(O_1, \ldots, O_{k-1}) - A_k) \cup D_k \end{aligned}$$

Theorem 3 provides us the missing link to the successful application of external breadth first search on planning graphs. Subtracting k previous layer *plus* the current layer from the successor list in an external breadth-first search guarantees its termination on finite planning graphs.

External Enforced Hill Climbing

Enforced Hill Climbing (EHC) is a *conservative* variant of hill climbing search. Starting from a start state, a breadth-first search is performed for a successor with a better heuristic value. As soon as such a successor is found, the hash tables are cleared and a fresh breadth-first search is started. The process continues until the goal is reached. Since EHC performs a complete breadth-first search on every state with a strictly better heuristic value, for directed graphs without dead-ends, Enforced Hill Climbing is complete and guaranteed to find a solution (Hoffmann & Nebel 2001).

Having external breadth-first search in hand, an external algorithm for enforced hill limbing can be constructed by utilizing the heuristic estimates and limiting the subtraction of previous layers to the locality of the graph as computed in the previous section. In Figure 4, we show the algorithm in pseudo-code format for external enforced hill-climbing. The externalization is embedded in the sub-procedure (Figure 3) that performs external breadth-first search for a state with better heuristic estimate.

As heuristic guidance, we chose relaxed plan heuristics for metric domains (Hoffmann 2003). The metric version of the propositional relaxed plan heuristic analyzes an extended layered plan graph, where each fact layer includes the encountered propositional atoms and numeric fluents. The forward construction of the plan graph iteratively applies operators until all goals are satisfied. The length of this relaxed plan is then used as a heuristic estimate to guide the search. The heuristic is neither admissible nor consistent, but very effective in practice.

In the worst case, External EHC performs a complete external breadth-first search for every improvement in the heuristic value starting from $h(\mathcal{I})$. This gives an I/O complexity of $O(h(\mathcal{I}) \cdot (sort(|E|) + locality \cdot scan(|V|)))$ I/Os. The term $sort(|E|)$ is due to external sorting the list of successors, while the second term is for subtracting the elements already expanded in the previous layers.

Plan-Reconstruction: In an internal non memory-limited setting, a plan is constructed by backtracking from the goal node to the start node. This is facilitated by saving with every node a pointer to its predecessor. For memory-limited frontier search, a divide-and-conquer solution reconstruction is needed for which certain relay layers have to be stored in main memory. In external search divide-and-conquer solution reconstruction and relay layers are not needed, since the exploration fully resides on disk.

There is one subtle problem: predecessor pointers are not available on disk. We propose to reconstruct plans by saving the predecessor together with every state and backtracking along the stored files while looking for matching predecessors. This results in an I/O complexity that is at most linear to the number of stored states. In the pseudo-codes, this procedure is denoted by *ConstructSolution*.

Cost-Optimal External BFS

In planning, we often have a monotone decreasing instead of a monotonic increasing cost function for the minimization problem. Consequently, we cannot prune states with an evaluation larger than the current one, and hence, are forced to look at all states.

Figure 5 shows the pseudo-code for external BFS incrementally improving an upper bound U on the solution length. The state sets that are used are represented in form of files. The search frontier denoting the current BFS layer is tested for an intersection with the goal, and this intersection is further reduced according to the already established bound. The I/O complexity of Cost-Optimal External BFS is $O(sort(|E|) + locality \cdot scan(|V|))$ I/Os.

Experimental Evaluation

In this section, we present two sets of experiments. All experiments are run on a Pentium-4 with 600 GB of hard-disk space and 2GB RAM running Linux. In the first set, we present non-optimal planning results with external enforced hill climbing. This set of experiments are performed on

Procedure Cost-Optimal-External-BFS

$U \leftarrow \infty$; $i \leftarrow 1$
Open$(-1) \leftarrow \emptyset$; *Open*$(0) \leftarrow \{\mathcal{I}\}$
while (*Open*$(i-1) \neq \emptyset$)
  $A(i) \leftarrow$ *Succ*(*Open*$(i-1)$)
  forall $v \in A(i)$
    if $v \in \mathcal{G}$ **and** *Metric*$(v) < U$
      $U \leftarrow$ *Metric*(v)
      ConstructSolution(v)
  $A'(i) \leftarrow$ *remove duplicates from* $A(i)$
  for *loc* $\leftarrow 1$ **to** *locality*
    $A'(i) \leftarrow A'(i) \setminus Open(i - loc)$
  Open$(i) \leftarrow A'(i)$
  $i \leftarrow i + 1$

Figure 5: Cost-Optimal External BFS; $\mathcal{I}$ is the start state, U represents the best solution cost found so far and $\mathcal{G}$ is the set of goal states.

one of the most challenging domains called Settlers, used in the third and the fourth international planning competitions. The distinguishing feature of the domain is that most of the domain semantics is encoded with numeric variables. The whole problem set for this domain has been solved by only one planner, SGPlan, which is based on goal-ordering and Lagrange optimization and finds plans very fast but with large plan lengths.

We have extended the state-of-the-art planning system Metric-FF for external exploration. Metric-FF uses helpful-action pruning to use the actions that are used in the relaxed-plan construction. This pruning destroys the completeness of the method but can be very effective in practice. Running external enforced hill climbing without helpful-action pruning resulted in shorter plan lengths while consuming lesser internal memory. Our plans are validated by VAL tool 3.2 (Howey, Long, & Fox 2005). In Table 1, we compare the plan lengths as found by external enforced hill climbing to the ones found by SGPlan. The first 5 problems can be solved by internal Enforced Hill Climbing too when using helpful-action pruning. All the experiments in this table are performed without using helpful-action pruning except for problem 3. We also report the internal memory and external space consumption by our algorithm. Note that the internal memory requirement can be scaled down to an arbitrary amount by using small internal buffers. The locality we encountered during this domain is 3 as observed by checking the duplicates in all previous layers for every BFS invoked.

The exploration for problem 7 was canceled in the middle because of time constraints. The total time taken was 3 days and 14 hours. It consumed 48.89 Gigabytes of external storage while internal process remained constant at 526 Megabytes. Starting from a heuristic estimate of 87, the algorithm climbed down to 8 with a total depth of 132. The bottleneck in this exploration is the computationally expensive calculations for the heuristic estimates. Figure 6 shows the histogram of states' distribution across different BFS layers for this problem. Here we see different groups

P. #	SGPlan	Ex-FF	Memory(MB)	Space(MB)
1	106	53	115	11
2	82	27	114	1
3	133	52	117	75
4	199	64	264	1,384
6	193	79	281	186
7	292	>132	526	>43,670

Table 1: Exploration Results on Settlers Domain.

of layers that actually correspond to starting a new external BFS when a state with a better heuristic value is found.

For the second set, we present optimal planning results on one of the hardest problem in the Traveling Purchase Problem (TPP) domain introduced for the fifth international planning competition. In order to avoid a conflict with the on-going competition, we only considered preferences p0A, p1A, p2A and p3A, in the goal condition. Table 2 shows the results of our exploration. We report the number of nodes in each layer obtained after refinement with respect to the previous layers. The locality as observed by the refinement with respect to all previous layers was 2. An entry in the *Goal Cost* column corresponds to the best goal cost found in that layer. The exploration had to be paused because of some technical problems in the machine making it unusable for two weeks. Total time consumed in exploration and duplicates removal is 29 days. The active layer being expanded is 23 with the best cost of 93 also found at this layer. The (*) in the 24-th row indicates that the reported node count and the space consumption for this layer does not reflect the final count. About 2×10^9 states have been generated. Space consumption lies by 450 Gigabytes taken by $858,535,767$ nodes residing on the harddisk. Time is largely consumed by intermediate duplicate detections, i.e., instead of waiting till a whole layer is generated, to save harddisk space. As is apparent from Table 2 that the branching factor has started to go down by a factor of 0.05 for each layer and a completion of this exploration is foreseen. The internal process size remained constant at 992 Megabytes. The exploration has been restarted using the pause-and-resume support implemented in the software.

Conclusions

Large graphs are often met in planning domains. Though inadmissible heuristics or some other technique to guide the search can result in faster search times, the plan lengths are often very large. We contribute two algorithms in this paper: Cost-optimal external breadth-first search and external enforced hill climbing search for metric planning. The crucial problem in external memory algorithms is the duplicate detection with respect to previous layers. Using the locality of the graph calculated directly from the operators themselves, we provide a bound on the number of previous layers that have to be looked at to avoid re-expansions.

We report the largest exploration with breadth-first search in planning domains - traversing a state space as large as about 500 Gigabytes in about 30 days. To the best of au-

BFS-Layer	Nodes	Space (GB)	Goal Cost
0	1	0.000000536	105
1	2	0.00000107	-
2	10	0.00000536	-
3	61	0.0000327	-
4	252	0.000137	-
5	945	0.000508	104
6	3,153	0.00169	-
7	9,509	0.00585	-
8	26,209	0.0146	103
9	66,705	0.0361	-
10	158,311	0.0859	-
11	353,182	0.190	101
12	745,960	0.401	-
13	1,500,173	0.805	-
14	2,886,261	1.550	97
15	5,331,550	2.863	-
16	9,481,864	5.091	-
17	16,266,810	8.735	96
18	26,958,236	14.476	-
19	43,199,526	23.197	-
20	66,984,109	35.968	95
21	100,553,730	53.994	-
22	146,495,022	78.663	-
23	205,973,535	110.601	93
24	231,540,651*	103.699*	-
SUM	858,535,767*	440.378*	93

Table 2: Exploration Results on Problem-5 of TPP Domain.

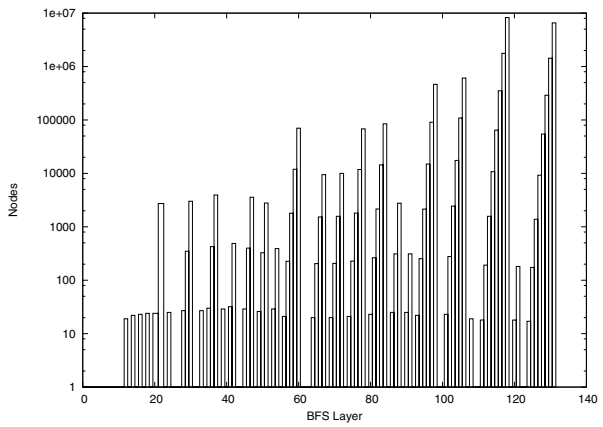

Figure 6: Histogram (logarithmic scale) on Number of Nodes in BFS Layers for External Enforced Hill-Climbing on Problem-7 of Settlers.

thors' knowledge, this is the longest running exploration reported in planning literature.

In future, we are also interested in exploring the idea of external beam-search, where only a set of best nodes are picked for the next iteration. Moreover, since states are kept on disk, external algorithms have a large potential for parallelization. We noticed that most of the execution time is consumed while calculating heuristic estimates. The approach we are currently working on, splits the layer that is being expanded, into several ones, and distributes the work among different processors. As states can be expanded independently of each other, an optimal speedup is foreseen.

References

Aggarwal, A., and Vitter, J. S. 1988. The input/output complexity of sorting and related problems. *Journal of the ACM* 31(9):1116–1127.

Arge, L.; Knudsen, M.; and Larsen, K. 1993. Sorting multisets and vectors in-place. In *Workshop on Algorithms and Data Structures (WADS)*, LNCS, 83–94.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *ICAPS*, To Appear.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predictable exogenous events. *JAIR* 25:187–231.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator sets. In *AIPS*, 150–158.

Helmert, M. 2002. Decidability and undecidability results for planning with numerical state variables. In *AIPS*, 303–312.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253 – 302.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291–341.

Howey, R.; Long, D.; and Fox, M. 2005. *Plan Validation and Mixed-Initiative Planning in Space Operations*, volume 117 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. chapter 6, 60–70.

Jabbar, S., and Edelkamp, S. 2006. Parallel external directed model checking with linear I/O. In *VMCAI*, 237–251.

Korf, R. E., and Schultze, P. 2005. Large-scale parallel breadth-first search. In *AAAI*, 1380–1385.

Korf, R. E., and Zhang, W. 2000. Divide-and-conquer frontier search applied to optimal sequence allignment. In *AAAI*, 910–916.

McDermott, D., et al. 1998. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Committee.

Munagala, K., and Ranade, A. 1999. I/O-complexity of graph algorithms. In *SODA*, 687–694.

Wah, B., and Chen, Y. 2006. Constrained partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence* 170(3):187–231.

Zhou, R., and Hansen, E. 2004. Structured duplicate detection in external-memory graph search. In *AAAI*. 683–689.

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