1F2-1

グラフ分割による並列探索の為の効率的な仕事分配手法

A Graph-Partitioning Based Approach for Parallel Best-First Search (Extended Abstract)[*1]

Yuu Jinnai　Alex Fukunaga

Graduate School of Arts and Sciences, The Univeristy of Tokyo

Parallel best-first search algorithms such as HDA* distribute work among the processes using a global hash function. We introduce an objective function to simultaneously minimize both search and communication overheads, which corresponds to partitioning the search space with a sparsest-cut objective. We propose SAZHDA*, which approximates this objective by partitioning the domain transition graph, an abstraction of the state space graph.

1. Introduction

The A* algorithm [Hart 68] is used in many areas of AI, including planning, scheduling, path-finding, and sequence alignment. Parallelization of A* can yield speedups as well as a way to overcome memory limitations – the aggregate memory available in a cluster can allow problems that can't be solved using 1 machine to be solved. Thus, designing scalable, parallel search algorithms is an important goal.

Hash Distributed A* (HDA*) is a parallel best-first search algorithm in which each processor executes A* using local OPEN/CLOSED lists, and generated nodes are assigned (sent) to processors according to a global hash function [Kishimoto 13]. HDA* can be used in distributed memory systems as well as multicore, shared memory machines, and has been shown to scale up to hundreds of cores with little search overhead. The performance of HDA* depends on the hash function used for assigning nodes to processors. Kishimoto et al. [Kishimoto 09, Kishimoto 13] showed that using the Zobrist hash function [Zobrist 70], HDA* could achieve good load balance and low search overhead. Burns et al [Burns 10] noted that Zobrist hashing incurs a heavy communication overhead because many nodes are assigned to processes that are different from their parents, and proposed AHDA*, which used an abstraction-based hash function originally designed for use with PSDD [Zhou 07] and PBNF [Burns 10]. Abstraction-based work distribution achieves low communication overhead, but at the cost of high search overhead. Abstract Zobrist hashing (AZH) [Jinnai 16a] achieves both low search overhead and communication overhead by incorporating the strengths of both Zobrist hashing and abstraction. While the Zobrist hash value of a state is computed by applying an incremental hash function to the set of features of a state, AZH first applies a feature projection function mapping features to abstract features, and the Zobrist hash value of the abstract features (instead of the raw features) is computed. Improvements to domain-independent, automated abstract feature generation methods for AZHDA* were proposed in [Jinnai 16a]. Although these methods seek to reduce search/communication overheads in the HDA* framework, these methods can be characterized as *bottom-up, ad hoc* approaches that introduce new mechanisms to address some particular problem within the HDA*/AZHDA* framework, but these methods do not allow *a priori* prediction of the communication and search overheads that will be incurred.

This paper proposes a new, *top-down* approach to minimizing overheads in parallel best-first search. Instead of addressing specific problems/limitations within the AZHDA* framework, we formulate an objective function which defines exactly what we seek in terms of minimizing both search and communications overheads, enabling a predictive model of these overheads. We then propose an algorithm which directly synthesizes a work distribution function approximating the optimal behavior according to this objective. The resulting algorithm, SAZHDA* significantly outperforms all previous variants of HDA*.

2. Background

Hash Distributed A* (HDA*) [Kishimoto 13] is a parallel A* algorithm where each processor has its own OPEN and CLOSED. A global hash function assigns a unique owner thread to every search node. Each thread T repeatedly executes the following: (1) For all new nodes n in T's message queue, if it is not in CLOSED (not a duplicate), put n in OPEN. (2) Expand node n with highest priority in OPEN. For every generated node c, compute hash value $H(c)$, and send c to the thread that owns $H(c)$.

Although an ideal parallel best-first search algorithm would achieve a n-fold speedup on n threads, several overheads can prevent HDA* from achieving linear speedup.

Communication Overhead (CO): Communication overhead is the ratio of nodes transferred to other threads: $CO := \frac{\text{\# nodes sent to other threads}}{\text{\# nodes generated}}$. CO is detrimental to performance because of delays due to message transfers (e.g., network communications), as well as access to data structure such as message queues. HDA* incurs communication overhead when transferring a node from the thread where it is generated to its owner according to the hash function. In general, CO increases with the number of threads. If nodes are assigned randomly to the threads, CO will be proportional to $1 - \frac{1}{\#thread}$.

Search Overhead (SO): Parallel search usually expands more nodes than sequential A*. In this paper we define search overhead as $SO := \frac{\text{\# nodes expanded in parallel}}{\text{\#nodes expanded in sequential search}} - 1$. SO can arise due to inefficient load balance (LB). If load balance is poor, a thread which is assigned more nodes than others will become a bottleneck – other threads spend their time expanding less promising nodes, resulting in search overhead.

[*1] This is an extended abstract paper of a paper which has been submitted to IJCAI.

There is a fundamental trade-off between CO and SO. Increasing communication can reduce search overhead at the cost of communication overhead, and vice-versa.

Zobrist Hashing, Abstraction, and Abstract Zobrist Hashing

In the original work on HDA*, Kishimoto et al. [Kishimoto 13] used Zobrist hashing [Zobrist 70]. The Zobrist hash value of a state s, $Z(s)$, is calculated as follows. For simplicity, assume that s is represented as an array of n propositions, $s = (x_0, x_1, ..., x_n)$. Let R be a table containing preinitialized random bit strings.

$$Z(s) := R[x_0]\ xor\ R[x_1]\ xor\ \cdots\ xor\ R[x_n]$$

Zobrist hashing seeks to distribute nodes uniformly among all threads, without any consideration of the neighborhood structure of the search space graph. As a consequence, communication overhead is high. Assume an ideal implementation that assigns nodes uniformly among threads. Every generated node is sent to another thread with probability $1 - \frac{1}{\#threads}$. Therefore, with 16 threads, $> 90\%$ of the nodes are sent to other threads, so communication costs are incurred for the vast majority of node generations.

Abstract Zobrist hashing (AZH) [Jinnai 16a] is a hybrid hashing strategy which augments the Zobrist hashing framework with the idea of projection from abstraction [Burns 10] to reduce CO. The AZH value of a state, $AZ(s)$ is:

$$AZ(s) := R[A(x_0)]\ xor\ R[A(x_1)]\ xor\ \cdots\ xor\ R[A(x_n)] \quad (1)$$

where A is a *feature projection function*, a many-to-one mapping from from each raw feature to an *abstract feature*, and R is a precomputed table for each abstract feature. Thus, AZH is a 2-level, hierarchical hash, where raw features are first projected to abstract features, and Zobrist hashing is applied to the abstract features.

Compared to Zobrist hashing, AZH incurs less CO due to abstract feature-based hashing. While Zobrist hashing assigns a hash value for each node independently, AZH assigns the same hash value for all nodes which share the same abstract features for all features, reducing the number of node transfers.

Domain-Independent Feature Projection Functions for Abstract Zobrist Hashing

The feature projection function plays a critical role in determining the performance of AZH, because AZH relies on the feature projection in order to reduce communications overhead. *Greedy abstract feature generation* (GreedyAFG) is a simple, domain-independent abstract feature generation method, which partitions each feature into 2 abstract features [Jinnai 16a]. GreedyAFG first identifies *atom groups* (sets of mutually exclusive propositions from which exactly one will be true for each reachable state, e.g., the values of a SAS+ multi-valued variable [Bäckström 95]). Each atom group G is partitioned into 2 abstract features S_1 and S_2, based G's undirected transition graph (nodes are propositions, edges are transitions), as follows: (1) assign the minimal degree node to S_1; (2) greedily add to S_1 the unassigned node which shares the most edges with nodes in S_1; (3) while $|S_1| < |G|/2$ repeat step (2) to guarantee ; (4) assign all unassigned nodes to S_2. This procedure guarantees $|S_2| \leq |S_1| \leq |S_2| + 1$.

3. Parallel Efficiency and Graph Partitioning

Although previous research on work distribution for HDA* proposed methods which reduce CO or SO, there was no explicit model which enabled the prediction of the actual efficiency achieved during search. In this section, we develop a metric to estimate the walltime efficiency as a function of CO and SO. First, we define *time efficiency* $\mathit{eff}_{actual} := \frac{speedup}{\#cores}$. Our ultimate goal is to maximize eff_{actual}.

Communication Efficiency: Assume that the communication cost between every pair of processors is identical. Then communication efficiency, the degradation of efficiency by communication cost, is $\mathit{eff}_c = \frac{1}{1+cCO}$, where $c = \frac{\text{time for sending a node}}{\text{time for generating a node}}$.

Search Efficiency: Assuming every core expands 1 node at a time and there are no idle cores, HDA* with p processes expands np nodes in the same wall-clock time A* requires to expand n nodes. Therefore, search efficiency, the degradation of efficiency by search overhead, is $\mathit{eff}_s = \frac{1}{1+SO}$.

Next, we discuss the relationship between SO and load balance (LB). It has been shown experimentally that an inefficient LB leads to high SO [Jinnai 16a], but there was no analysis on *how* LB leads to SO. Assume that all nodes have unique f values and f^* to be an optimal cost. We define *state space* S as $S(v \in S | f(v) \leq f^*)$. To guarantee the optimality of a solution, HDA* needs to expand all nodes in S. Assume that the number of duplicate nodes is negligible, and that a work distribution method allocates nodes in $S_i(v \in S_i | v \in S; v \notin S_j; i \neq j)$ to process p_i. As HDA* needs to expand all nodes in state space, each process expands $\max_j |S_j|$ nodes before HDA* terminates. As a consequence, process p_i expands $\max_j |S_j| - |S_i|$ nodes outside the search space ($f > f^*$), which results in SO. The sum for all processes is $SO = \sum_i^p (\max_j |S_j| - |S_i|) = p(LB - 1)$, where $LB = \frac{\max_j |S_j|}{avg_i |S_i|}$, and $avg_i |S_i|$ is average $|S_i|$ of all processes.

Using CO and LB, we can estimate the time efficiency eff_{actual}. eff_{actual} is proportional to the product of communication and search efficiency: $\mathit{eff}_{actual} \propto \mathit{eff}_c \cdot \mathit{eff}_s$. There are overheads other than CO and SO such as hardware overhead (i.e. memory bus contention) that affect performance [Burns 10], but we assume that CO and SO are the dominant factors in determining efficiency.

We define *estimated efficiency* $\mathit{eff}_{esti} := \mathit{eff}_c \cdot \mathit{eff}_s$, and we use this metric to estimate the actual performance (efficiency) of a work distribution method.

$$\begin{aligned} \mathit{eff}_{esti} &= \mathit{eff}_c \cdot \mathit{eff}_s = 1/\big(\ (1 + cCO)(1 + SO)\ \big) \\ &= 1/\big(\ (1 + cCO)(1 + p(LB - 1))\ \big) \qquad (2) \end{aligned}$$

3.1 Work Distribution as a Graph Partitioning

Work distribution methods for hash-based parallel search distribute nodes by assigning a process to each node in the state space. Our goal is to design a work distribution method which maximize efficiency by reducing LB and CO. The workload distribution of a parallel search method can be modeled as a partitioning of a *workload graph* WG which is isomorphic to the search space graph, i.e., nodes in WG correspond to states in the search space, and edges in the workload graph correspond to edges in the search space. The distribution of nodes among p processors is a p-way partition of WG, where nodes in partition S_i are assigned to process p_i. *Given a partitioning of WG, LB and CO can be predicted directly from the structure of the graph, without having to run HDA* and measure LB and CO experimentally, i.e., it is possible to analyze the efficiency of a workload distribution method independently from its parallel execution.* LB corresponds to load balance of the partitions and CO is the # of edges between partitions over the number of total edges, i.e.,

$$CO = \frac{\sum_i^k \sum_{j>i}^k E(S_i, S_j)}{\sum_i^k \sum_{j>i}^k E(S_i, S_j)}, \quad LB = \frac{\max_i |S_i|}{avg_i |S_i|} \tag{3}$$

where $|S_i|$ is the # of nodes in partition S_i and $E(S_i, S_j)$ is the # of edges between S_i and S_j. Applying the equation to nodes with $f < f^*$, gives us the values for CO, LB, SO, and eff_{esti} of the work distribution method for the instance.

We compared eff_{esti} model and actual efficiency on planning instances and the results are shown in Figure 1a. Using least-square regression to estimate the coefficient a in $\mathit{eff}_{actual} = a \cdot \mathit{eff}_{esti}$, $a = 0.86$ with variance of residuals 0.013. Note that $a < 1.0$ because there are other sources of overhead which not accounted for in eff_{esti}, (e.g. memory bus contention) which affect performance [Burns 10].

4. Sparsity-Based Workload Partitioning

A standard approach to workload balancing in parallel scientific computing is graph partitioning, where the workload is represented as a graph, and a partitioning of the graph according to some objective (usually the cut-edge ratio metric) represents the allocation of the workload among the processors [Hendrickson 00, Buluç 13].

In Sec. 3., we showed that eff_{esti} can be used to effectively predict the actual efficiency of a work distribution. By defining a graph cut objective such that the partitioning the nodes in the search space (with $f < f^*$) according to this graph cut objective corresponds to maximizing eff_{esti}, we would have a method of generating an optimal workload distribution.

A *sparsest cut* objective for graph partitioning problem seeks to maximize the *sparsity* of the graph [Leighton 99].We define sparsity as

$$Sparsity := \frac{\prod_i^k |S_i|}{\sum_i^k \sum_{j>i}^k E(S_i, S_j)}, \tag{4}$$

where $|S_i|$ is the sum of nodes weights in partition S_i, $E(S_i, S_j)$ is the sum of edge weights between partition S_i and S_j. Consider the relationship between the sparsity of a state space graph for a search problem and the eff_{esti} metric defined in the previous section. By equations 2 and 3, Sparsity simultaneously considers both LB and CO, as the numerator $\prod_i^k |S_i|$ corresponds to LB and the denominator $\sum_i^k \sum_{j>i}^k E(S_i, S_j)$ corresponds to CO.

The comparison of Sparsity and eff_{esti} using METIS graph partitioning package is shown in Figure 1c. There is a clear correlation between sparsity and eff_{esti}. Thus, partitioning a graph to maximize $sparsity$ should maximize the eff_{esti} objective, which should in turn maximize actual walltime efficiency.

5. Sparsest Cut Abstract Feature Generation

Since eff_{esti} model accurately estimates actual efficiency, and sparsity has a strong correlation with eff_{esti}, a partition of the state space graph which minimize sparsity should be a (near) optimal work distribution which maximizes eff_{esti}. Unfortunately, it is impractical to directly apply standard graph partitioning algorithms to the state space graph because the state space graph is a huge *implicit* graph, and the partitioner needs as input the explicit representation of the relevant state space graph (a solution to the search problem itself!).

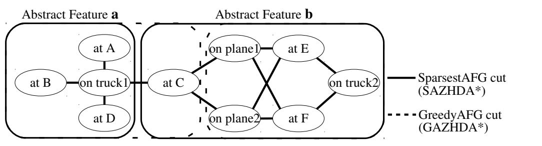


Figure 2: Example of sparsest cut and GreedyAFG to a domain transition graph in logistic domain. Assume unit edge cost for simplicity, SparsestAFG achieves $Sparsity = 24$.

Therefore, to generate a work distribution method for parallel A*, we have to partition some graph which is easily accessible from the domain description (e.g. PDDL, SAS+). We propose *Sparsest-cut based Abstract Zobrist HDA** (**SAZHDA***), which approximates the optimal strategy by partitioning *domain transition graphs*.

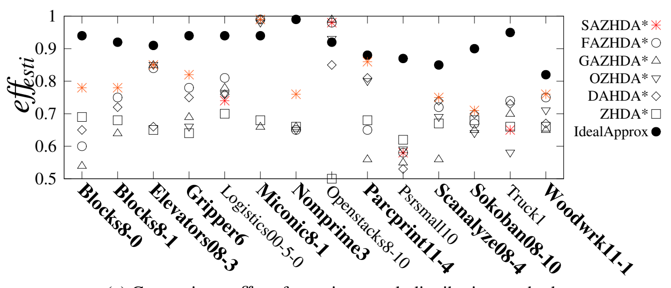
Given an atom group $x \in X$, its domain transition graph (DTG) $\mathcal{D}_x(E, V)$ is a directed graph where vertices V corresponds to the value of the atom group and edges E to their transitions, where $(v, v') \in E$ if and only if there is an operator o with $v \in del(o)$ and $v' \in add(o)$ [Jonsson 98]. We used DTGs of SAS+ variables. Figure 2 shows the sparsest cut of a DTG (for the variable representing package location) in the standard `logistics` domain. Maximizing sparsity results in cutting only 1 edge (i.e., good load balance).

SAZHDA* treats each partition of the DTG as an abstract feature in the AZH framework, assigning a hash value to each abstract feature. Since the AZH value of a state is the XOR of the hash values of the abstract features (Eqn 1), 2 nodes in the state space are in different partitions if and only if they are partitioned in *any* of the DTGs. . Therefore, SAZHDA* generates 2^n partitions from n DTGs, which are then projected to the p processors (by taking the partition ID modulo p). To make it likely that partitioning over the DTGs is a good approximation for partitioning the actual state space graph, we set a weight for each edge $e = \frac{\text{\# ground actions which correspond to the transition}}{\text{\# ground actions}}$. As DTGs typically have < 10 nodes, we compute the optimal sparsest cut with a straightforward branch-and-bound procedure.

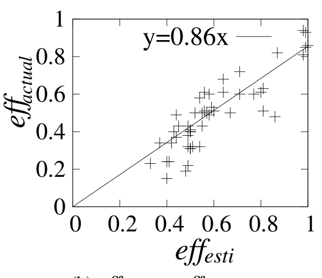
6. Evaluation of SAZHDA*

Instance	A*		SAZHDA*	FAZHDA*	GAZHDA*	OZHDA*	DAHDA*	ZHDA*
	time	expd	speedup	speedup	speedup	speedup	speedup	speedup
Blocks10-0	129.26	11065451	**27.16**	26.01	21.80	16.46	24.44	14.93
Blocks11-1	621.74	52736900	**34.37**	34.25	29.20	28.59	33.22	27.98
Elevators08-5	165.22	7620122	27.65	**29.35**	17.57	22.57	27.96	27.68
Elevators08-6	453.21	18632725	**44.12**	31.71	31.37	41.12	27.33	21.88
Gripper8	517.41	50068801	26.67	**29.45**	21.86	24.77	22.14	21.66
Logistics00-10-1	559.45	38720710	**27.64**	25.98	19.77	27.40	26.88	19.40
Miconic11-0	232.07	12704945	**42.83**	42.43	22.10	34.96	41.98	9.05
Nomprime5	309.14	4160971	**28.89**	22.87	18.55	16.66	18.89	17.85
Openstacks08-21	554.63	19901601	**46.03**	29.97	40.66	39.34	23.42	39.06
PipesNoTk10	157.31	2991859	15.73	15.64	15.58	15.22	**17.12**	14.88
Scanalyzer08-6	195.49	10202667	**32.92**	31.23	20.28	23.70	24.81	19.38
Total time[sec]	3894.93	228806752	**122.84s**	135.96s	163.58s	147.12s	153.32s	188.11s
Average speedup	354.08	20800614	**32.18**	28.99	23.52	26.44	26.20	21.25
Average eff_{actual}			**0.67**	0.60	0.49	0.55	0.55	0.44
Average eff_{esti}			**0.64**	0.57	0.56	0.57	0.56	0.47
Average CO			0.44	0.50	0.78	0.61	0.40	0.97
Average SO			0.11	0.13	0.01	0.13	0.28	0.07

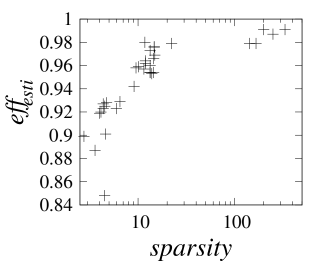
Table 1: Comparison of speedups, search/communication overheads (SO, CO), estimated/actual efficiencies (eff_{esti}, eff_{actual}).



(a) Comparison eff_{esti} for various work distribution methods



(b) eff_{esti} vs. eff_{actual}



(c) $sparsity$ vs. eff_{esti}

Figure 1: Figure 1a compares eff_{esti} when $c = 1.0$, $p = 48$. **Bold** indicates that SAZHDA* has the best eff_{esti} (except for IdealApprox). Figure 1b compares eff_{esti} and the actual experimental efficiency when $c = 1.0$, $p = 48$. $eff_{actual} = 0.86 \cdot eff_{esti}$ with variance of residuals = 0.013 (least-squares regression). Figure 1c compares sparsity vs. eff_{esti}. For each instance, we generated 3 different partitions using METIS with load balancing constraints which force METIS to balance randomly selected nodes, to see how degraded sparsity affects eff_{esti}.

Figure 1a shows eff_{esti} for the various work distribution methods, including SAZHDA*. To evaluate how these methods compare to an ideal (but impractical) model which actually applies graph partitioning to the entire search space (instead of partitioning DTG as done by SAZHDA*), we also evaluated *IdealApprox*, a model which partitions the entire state space graph using the METIS (approximate) graph partitioner [Karypis 98]. IdealApprox first enumerates a graph containing all nodes with $f \leq f^*$ and edges between these nodes and ran METIS with the sparsity objective (Eqn. 4) to generate the partition for the work distribution. Generating the input graph for METIS takes an enormous amount of time (much longer than the search itself), so IdealApprox is clearly an impractical model, but it is a useful approximation for an ideal work distribution.

Not surprisingly, IdealApprox has the highest eff_{esti}, but among all of the practical methods, SAZHDA* has the highest eff_{esti} overall. As we saw that eff_{esti} is a good estimate of actual efficiency, the result suggest that SAZHDA* outperforms other methods. In fact, as shown in Table 1, SAZHDA* achieved a good balance between CO and SO and had the highest actual speedup overall, significantly outperforming all other previous methods including previous state-of-the-art FAZHDA* [Jinnai 16b].

7. Conclusions

We proposed and evaluated a new, domain-independent approach to work distribution for parallel best-first search in the HDA* framework. The main contributions are (1) proposal and validation of eff_{esti}, a model of search and communication overheads for HDA* which can be used to predict actual walltime efficiency, (2) formulating the optimization of eff_{esti} as a graph partitioning problem with a sparsity objective, and validating the relationship between eff_{esti} and the sparsity objective, and (3) SAZHDA*, a new work distribution method which approximate the optimal strategy by partitioning domain transition graphs. We experimentally showed that SAZHDA* significantly improves both estimated efficiency (eff_{esti}) as well as actual performance (walltime efficiency) compared to previous work distribution methods. Our results demonstrate the viability of approximating the partitioning of the entire search space by applying graph partitioning to an abstraction of the state space (i.e., the DTG).

References

[Bäckström 95] Bäckström, C. and Nebel, B.: Complexity results for SAS+ planning, *Computational Intelligence*, Vol. 11, No. 4, pp. 625–655 (1995)

[Buluç 13] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C.: Recent advances in graph partitioning, *Preprint* (2013)

[Burns 10] Burns, E. A., Lemons, S., Ruml, W., and Zhou, R.: Best-First Heuristic Search for Multicore Machines, *Journal of Artificial Intelligence Research*, Vol. 39, pp. 689–743 (2010)

[Hart 68] Hart, P. E., Nilsson, N. J., and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107 (1968)

[Hendrickson 00] Hendrickson, B. and Kolda, T. G.: Graph partitioning models for parallel computing, *Parallel computing*, Vol. 26, No. 12, pp. 1519–1534 (2000)

[Jinnai 16a] Jinnai, Y. and Fukunaga, A.: Abstract Zobrist Hash: An Efficient Work Distribution Method for Parallel Best-First Search, in *Proc. AAAI* (2016)

[Jinnai 16b] Jinnai, Y. and Fukunaga, A.: Automated Creation of Efficient Work Distribution Functions for Parallel Best-First Search, in *Proc. ICAPS* (2016)

[Jonsson 98] Jonsson, P. and Bäckström, C.: State-variable planning under structural restrictions: Algorithms and complexity, *Artificial Intelligence*, Vol. 100, No. 1, pp. 125–176 (1998)

[Karypis 98] Karypis, G. and Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on scientific Computing*, Vol. 20, No. 1, pp. 359–392 (1998)

[Kishimoto 09] Kishimoto, A., Fukunaga, A. S., and Botea, A.: Scalable, Parallel Best-First Search for Optimal Sequential Planning, in *Proc. ICAPS*, pp. 201–208 (2009)

[Kishimoto 13] Kishimoto, A., Fukunaga, A., and Botea, A.: Evaluation of a simple, scalable, parallel best-first search strategy, *Artificial Intelligence*, Vol. 195, pp. 222–248 (2013)

[Leighton 99] Leighton, T. and Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms, *Journal of the ACM (JACM)*, Vol. 46, No. 6, pp. 787–832 (1999)

[Zhou 07] Zhou, R. and Hansen, E. A.: Parallel Structured Duplicate Detection, in *Proc. AAAI*, pp. 1217–1223 (2007)

[Zobrist 70] Zobrist, A. L.: A new hashing method with application for game playing, *reprinted in International Computer Chess Association Journal (ICCA)*, Vol. 13, No. 2, pp. 69–73 (1970)