Pathfinding in Games

Adi Botea[1], Bruno Bouzy[2], Michael Buro[3], Christian Bauckhage[4], and Dana Nau[5]

1 IBM Research, Dublin, Ireland
adibotea@ie.ibm.com
2 Université Paris Descartes, France
bruno.bouzy@parisdescartes.fr
3 University of Alberta, Canada
mburo@cs.ualberta.ca
4 Fraunhofer IAIS, 53754 Sankt Augustin, Germany
christian.bauckhage@iais.fraunhofer.de
5 University of Maryland, USA
nau@cs.umd.edu

Abstract

Commercial games can be an excellent testbed to artificial intelligence (AI) research, being a middle ground between synthetic, highly abstracted academic benchmarks, and more intricate problems from real life. Among the many AI techniques and problems relevant to games, such as learning, planning, and natural language processing, pathfinding stands out as one of the most common applications of AI research to games. In this document we survey recent work in pathfinding in games. Then we identify some challenges and potential directions for future work. This chapter summarizes the discussions held in the pathfinding workgroup.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases path finding, search, games

Digital Object Identifier 10.4230/DFU.Vol6.12191.21

1 Introduction

Pathfinding is an important application of artificial intelligence to commercial games. Despite a significant progress seen in recent years, the problem continues to constantly attract researchers' attention, which is explained in part by the high performance demands that pathfinding implementations have to satisfy in a game. Pathfinding engines are allocated limited CPU and memory resources. Moves have to be computed in real time, and often there are many mobile units to compute paths for. Paths have to look sufficiently realistic to a user. Besides a more standard, single-agent pathfinding search on a fully known, static map, games feature more complicated variations of the problem, such as multi-agent pathfinding, adversarial pathfinding, dynamic changes in the environment, heterogeneous terrains and mobile units, incomplete information, and combinations of these.

This chapter looks at games pathfinding research, with an emphasis towards the future. It summarizes the discussions held in the Pathfinding Workgroup at the Dagstuhl Seminar on Computational and Artificial Intelligence in Games. A shorter summary of the group's findings is available as part of a report on the entire seminar [24]. The first half of this document overviews recent progress in the area. Then, current challenges and future work directions are discussed.

2 Survey of Pathfinding Methods

We start by indicating a popular, baseline approach to pathfinding. The three key elements of the baseline approach are a graph representation of a map, a search algorithm, and a heuristic function to guide the search. Grid maps are a popular way of discretizing a game map into a search graph. As part of the process, a map is partitioned into atomic square cells, sometimes called tiles. Depending on the map initial topology, a tile is marked as either traversable or blocked. A mobile unit can occupy one traversable tile at a time. Traversable tiles become nodes in the search graph. Graph edges connect adjacent traversable tiles. Depending on the grid type, adjacencies are defined in the 4 cardinal directions (4-connected grid maps), or in the 4 cardinal + 4 diagonal directions (8-connected grid maps).

According to Björnsson et al. [1], the A* [16] algorithm used to be the de facto standard in pathfinding search. A* is a well known best-first search method, used in many other search problems, including AI planning and puzzle solving, to name just a few. To speed up a search, A* uses a heuristic function $h(n)$, which estimates the distance from a given node n to a goal node. Heuristics that do not overestimate the true distance to a goal are said to be admissible. A* with an admissible heuristic returns optimal solutions. On 4-connected maps, the Manhattan distance is a well known admissible heuristic. Given two nodes with coordinates (x_1, y_1) and (x_2, y_2), the Manhattan distance is defined as $\Delta x + \Delta y$, where $\Delta x = |x_1 - x_2|$ and $\Delta y = |y_1 - y_2|$. On a map with no obstacles, the Manhattan distance coincides with the true distance. The Octile distance is a variant of the Manhattan distance adapted to 8-connected maps. It is defined as $\sqrt{2} \times m + (M - m)$, where $m = \min(\Delta x, \Delta y)$ and $M = \max(\Delta x, \Delta y)$.

Part of the work overviewed in the rest of this section presents extensions of the baseline approach, being based on online search. Extensions include using more informative heuristic methods, replacing a low-level grid map with a hierarchy of smaller search graphs, exploiting symmetries to prune the search space in A* search, and using triangulation map decomposition instead of grid maps.

Other contributions eliminate graph search at runtime. For example, compressed path databases rely on a database of pre-computed all-pairs shortest path (APSP) data. Moves are retrieved from a compressed APSP table. This is suitable to games where all potential paths are precomputed, to largely free the CPU from the burden of path computation during game play. The data compression helps reducing the memory overhead associated with caching the precomputed paths.

2.1 Beyond Manhattan: Modern Pathfinding Heuristics on Grid Maps

The Manhattan and the Octile heuristics have the advantage of being simple, fast to compute, and reasonably accurate on many map topologies. However, more informed heuristics can reduce the number of states expanded (or visited) in a search. In this section we overview other heuristics in pathfinding, which can achieve a better accurracy, depending on factors such as the map topology. They use pre-processing and additional memory to cache the results of the pre-processing.

Goldberg and Harrelson [9] describe an admissible heuristic as part of their ALT search method. Consider a given node l, called a landmark, on the graph. A pre-computation step provides a look-up table with true distances $d(n, l)$ from every node n to l. For simplicity, assume an undirected graph. Given two arbitrary nodes n_1 and n_2, $|d(n_1, l) - d(n_2, l)|$ is an admissible estimation of the true distance from n_1 to n_2. The ALT admissible heuristic takes the maximum $\max_l |d(n_1, l) - d(n_2, l)|$ over a subset of landmarks. Goldberg and Harrelson applied their ideas to graphs such as roadmaps and grids with non-uniform costs. The heuristic has independently been developped by other researchers, including Sturtevant et al. [32], who call it the differential heuristic. Goldenbert et al. [11] show how the differential heuristic can be compressed. As an example of when the ALT heuristic is more informative than the Manhattan heuristic, we note that ALT can potentially identify unreachable locations (∞ heuristic value), whereas Manhattan might return a finite value.

ALTBest$_P$ [6] is a variant of the ALT heuristic that uses one single landmark, from a pre-defined subset of P landmarks. Specifically, at the beginning of a search, a landmark is selected that gives the highest heuristic distance for the initial state: $\arg\max_l |d(s,l) - d(g,l)|$, where s and g are the start and the goal. Furthermore, at each evaluated node, the heuristic given by the selected landmark is compared with the Manhattan heuristic, and their maximum is used as a heuristic evaluation.

Björnsson and Halldórsson [2] introduce the *dead-end heuristic* and the *gateway heuristic*, two methods that use an automated decomposition of the map into disjoint areas. The dead-end heuristic identifies areas that are irrelevant when solving a given instance, and assign a heuristic value of ∞ to all locations contained in such areas. The gateway heuristic identifies gateway points between adjacent areas in the partitioning, and pre-computes a table with optimal distances between pairs of entrances. Thus, additional memory, with a size quadratic to the number of gateways, is traded for an improved heuristic accuracy. A similar trade-off is taken by Sturtevant et al. [32], with their *canonical heuristic*, and Goldenberg et al. [10], with the *portal heuristic*. Sturtevant et al. [32] use the generic name of *true distance heuristics* for the family of heuristics that store exact distances between pairs of selected nodes.

2.2 Hierarchical Abstraction on Grid Maps

Hierarchical abstraction can potentially speed up a search by replacing it with a series of smaller searches. Abstract solutions computed in a higher-level state space representation are gradually refined, possibly through several hierarchical levels, down to the ground level of the original search space.

Pathfinding is an application where hierarchical abstraction has been successful. Hierarchical pathfinding methods, such as work mentioned in this section, can ensure that an initial abstract solution can be refined into a low-level solution. This eliminates the need to backtrack across hierarchical levels, thus eliminating a source of a potentially great slow down.

HPA* [3] decomposes a map into disjoint square sectors. Entrance points between adjacent sectors are identified and added as nodes into an abstracted search space. Abstract edges connect pairs of entrance points placed on the border of the same sector. In effect, in the abstracted space, a move traverses one sector in one step. Additional abstracted edges connect the start node (and the target) to the entrance points of its sector. An abstract solution contains macro moves such as sector-traversing moves. In a refinement step, a search restricted to the area of one sector converts a macro step into a sequence of actual moves. The method can have more than 2 hierarchical levels.

Partial Refinement A* (PRA* [34]) combines hierarchical abstraction with fast partial path refinement to effectively interleave path planning with path execution, which is relevant to robotics and video games. PRA* first builds a hierarchical map respresentation bottom-up by mapping small connected regions (e.g., traversable 2x2 tile squares) to tiles on the next level of abstraction while preserving connectivity. This greedy abstraction process creates pyramids of coarser and coarser map representations up to the point in which only single tiles remain that represent the original connected components. For finding paths subject to given time and path quality constraints, we choose an abstraction layer and find nodes representing the start and endpoints on the original map. Starting at this level, PRA*(k) uses A* to compute the shortest path and then projects the first k steps down to the next map abstraction level, where the shortest path is determined by only considering a small corridor around the projected path, etc. This process quickly finds high-quality initial paths on the original map which allows objects to start moving in the right direction quickly. Later, Sturtevant [33] described a hybrid approach that combines ideas of HPA* and PRA* to decrease memory used for abstraction.

HAA* [12] extends HPA* in two directions. First off, in classical pathfinding, a map is partinioned into traversable terrain and blocked areas. HAA* makes a finer distinction, allowing to define several types of terrain, such as ground, water, and walls. Secondly, HAA* handles mobile units with variable sizes and variable terrain traversal capabilities.

The idea of splitting a map into disjoint rectangular blocks is also exploited in Block A* [39]. Given a pre-determined block size $m \times n$, Block A* pre-computes a database of all possible block topologies (i.e., all possible combinations of traversable and blocked grid cells). For each block, optimal distances between pairs of boundary cells are stored. At runtime, Block A* expands an entire block at a time, as opposed to one individual cell, as it is the case in regular A*. Block A* is adapted to perform any-angle pathfinding [8], which, unlike standard grid-based pathfinding, is not limited to the 8 major compass directions.

While the map decomposition used in HPA*, HAA* and Block A* explicitly assumes a gridmap encoding of the underlying topology, it seems that they could in principle be extended to other graphs with an (almost) planar structure. One possible approach would be to slice the bounding rectangle of the graph with square or rectangular blocks, obtaining the map decomposition. At the same time, PRA* appears to be more straightforward to apply to non-gridmap graphs, as its core abstraction method relies on identifying cliques.

2.3 Symmetry Elimination

The frequently used assumption that traversable areas on a map have a uniform traversal cost leads to a considerable degree of symmetry in a pathfinding problem. Consider a path encoding where each move is represented as a direction. For example, on a 4-connected gridmap, a path from A to B that goes three steps to the left and three steps upwards would be encoded as *llluuu*. Assumming there are no obstacles in the grid-aligned rectangle having A and B as opposite corners, there are many equivalent paths from A to B, obtained by interleaving the moves: *lluluu*, *lluulu* and so on. This can lead to an excessive, useless exploration of some parts of a map.

Symmetry reduction can be based on empty rectangles identified on a map [13, 14]. Given two nodes A and B on the border of an empty rectangle, all optimal paths connecting these are symmetrical to each other, and considering only one such a path is sufficient. Thus, search can be sped up by defining macro-operators to cross rectangles without visiting any interior nodes. Jump point search [15] takes a different approach to symmetry elimination. It allows pruning part of the successors of a node, ensuring that optimality and completeness are preserved.

2.4 Triangulation-Based Pathfinding

Grid-based map representations are common in pathfinding applications such as video games, because of their conceptual simplicity that leads to straight-forward and fast implementations. However, there are several drawbacks that may render grid-based representations less useful in applications in which world objects don't have uniform sizes or shapes, they are not axis-aligned, or they can move in arbitrary directions. Another problem is that basic grid-based representations waste space in case large map regions are empty. To address these problems Demyen and Buro [7] present two pathfinding algorithms TA* (Triangulation A*) and TRA* (Triangulation Reduction A*). TA* uses dynamic constrained Delaunay triangulation (DCDT [18]) to represent maps in which obstacles are defined by polygons, and finds optimal any-angle paths for circular objects by running an A* like algorithm on graphs induced by the triangulation.

TRA* improves on TA* by applying a topological map abstraction that contracts each corridor in the triangulation graph to a single edge and tree components in which pathfinding is trivial are removed and handled separately. What is left is a graph only containing degree-3 nodes. Its size is proportional to the number of obstacles on the map, which can be considerably smaller than the triangulation graph. Because of this size reduction, TRA* runs faster than TA* and much faster than A* on common game maps while generating high-quality paths. With the geometric map represenation the mentioned restrictions of grid-based approaches are lifted. A TA* variant is being used in the StarCraft 2 game engine.

2.5 Real-Time Search

Real-time search [20] works under the assumption that an agent has a limited amount of time before having to make a move. Many researchers have addressed the problem since Korf's original work [20]. The general solving framework includes a planning step, a learning step and an acting step. The cycle repeats until the agent reaches the target. In the planning step, the agent explores the area around its current location. Learning uses the results of the exploration to update knowledge such as heuristic estimations of the current state or other states. Acting refers to making one or several moves, changing the current state of the agent. The many real-time algorithms available differ in the exploration strategy (e.g., A* search vs breadth first search), how to perform the heuristic update, and what state to select as the next current state.

For a long time, real-time search has not been considered as a viable approach to be implemented in a game. A main cause is the phenomenon of *scrubbing*: when real-time search reaches a plateau, an agent has to slowly learn more accurate heuristic values for the states in the plateau, before being able to escape. This results in an irrational-looking moving pattern of the agent, moving around within a local area for a long time. Solution paths can be very long, and so can be the total time to reach the target.

Recently, a line of research on real-time search has focused on eliminating, either completely or to a great extent, the learning step and thus the scrubbing behaviour [5, 23]. A key insight that contributes to the success of such methods is the idea of using *hill-climbing moves*. These are moves that look the most promising after searching forward only one step ahead, i.e., evaluating only the successors of a current state. Often, two nodes can be connected by performing solely hill-climbing moves. For example, consider two nodes within an area with no obstacles. When a path can be decomposed into subgoals such that the next sub-goal can be reached with hill-climbing moves, moves are computed fast, with no learning and scrubbing necessary.

2.6 Compressed Path Databases

Most pathfinding methods search at the path query time to compute a path on demand. An alternative approach is to perform path precomputation and lookup [36], returning paths orders of magnitude faster than online search. Straightforward ways of storing precomputed paths have prohibitive memory requirements, which has been a major impediment to adopting path lookup on a larger scale. Recently, a structure called a compressed path database [4] has been shown to achieve a massive lossless compression, bringing memory needs down to reasonable values for many grid maps. Compressed path databases exploit an idea previously observed and used in [27], with no application to games. In many cases, the first move on an optimal path, from a start location s to any target t in a contiguous remote area, is the same. The original authors call this property path coherence. For a start–target pair (s, t), let the first-move label of t identify an optimal move to take in s towards t. Compression involves finding, for a given s, a contiguous area A where all the targets $t \in A$ have the same first move label. This allows to replace many identical first-move labels, one for each (s, t) pair, with one single record for the pair (s, A).

Advantages of compressed path databases include the speed of path retrieval and the optimality of solutions. The first-move lag (i.e., the time to wait until the first move is performed) is very low, since each move is retrieved independently from the subsequent moves on the path. At the same time, compressed path databases pose a number of challenges to be addressed in the future, as we discuss later in this report.

2.7 Multi-Agent Pathfinding

Multi-agent pathfinding addresses the problem of finding paths for a set of agents going to their goals. Each agent may have its own goal or all the agents may have a global goal. At each timestep, all agents move according to their plans. The set of agents has to find the minimal cost for reaching the set of goals, or to maximize some quantity. The cost can be the elapsed time. Compared to mono-agent pathfinding, two obstacles appear clearly. The first one is the size of the set of (joint) actions which is exponential in the number of agents, which lead to difficulty for a global planning method. And when considering the elementary solutions to each mono-agent problem in order to find a global solution, the second one is managing collisions between agents.

Furthermore, multi-agent pathfinding can be cooperative, when all the agents help each other and try to optimize in the same direction. But it can be adversarial when an army tries to reach a point and another one prevents this from happening.

Starting points for cooperative pathfindind (CPF) are optimal methods like centralized A*. They work well on small problems. To avoid the exponential number of actions in the number of agents, Standley proposes the mechanism of operator decomposition (OD) [30]. Instead of considering that the set of agents decides its joint action (operator), the agents decide their elementary action in turn, one after each other. With some care to special cases, A* associated with OD and a perfect heuristic is admissible and complete [30]. Another idea for optimal solutions is independence detection (ID) [31]. Each agent plans its optimal path to the goal with A*. Then conflicts are detected, and resolved if possible. When a conflict between two agents is not solvable, the two agents are gathered into a group, and an optimal solution is found for this group. Conflict between groups are detected and so forth. When few conflicts occur this method avoids the exponential size of the set of actions, but in the worst case, this method is inferior to any centralized method.

The Increasing Cost Tree Search (ICTS) [28] is a two-level search: a global level and a low level. At the global level, the search proceeds on an Incremental Cost Tree (ICT) in which one node corresponds to a vector of costs. The global cost is the sum of individual cost heuristic (SIC). At the root of the ICT the costs are the optimal costs of agents considered alone, and the global cost cannot be smaller. At depth δ, the sum of the costs of a node is the optimal cost plus δ. A node is a goal for ICT iff there is an effective combination of individual paths without conflict and with costs corresponding to the costs of the node. ICT search is a breadth-first search for a goal node that progressively increases the best possible optimal cost.

In video games, optimal complete methods are not always searched because they are not scalable. Instead, suboptimal, incomplete but tractable methods exist. Cooperative A* [29] decomposes the whole task into a series of single agent searches in a three-dimensional space including time. A wait move is added to the set of actions of the agents. The computed routes are stored in a reservation table that forbids to use its entries in future searches. Hierarchical cooperative A* (HCA*) uses the idea of Hierarchical A* [17] with the simplest possible hierarchy: the domain with all agents removed. An issue of HCA* is how to terminate: sometimes an agent sitting on its destination must move to give the way to another agent. Another issue is the ordering of agents, and the most important one is computing complete routes in the 3-dimensional space. Therefore, Windowed HCA* (WHCA*) simply limits the lookahead in each search it performs to a fixed-horizon time window [29].

Multi-agent pathfinding can be solved suboptimally in low-polynomial time [21, 26]. However, the practical performance of the available complete method is an open question. Recent research has lead to the creation of multiple suboptimal methods that are complete on well-specified subclasses of instances. Push-and-Swap [25] works on problems with at least two free nodes (i.e., problems where the number of nodes in the graph representing the map exceeds the number of mobile units by at least two). MAPP [38] works on problems of the Slidable class. TASS [19] works on problems with at least four free nodes. The TASS method is a tree-based agent swapping strategy. There are other sub-optimal methods that work well in practice, but whose completeness range has not been characterized [29, 35, 37].

3 Looking at the Future

In this section we discuss a number of current challenges and directions for the future.

3.1 Bridging the Industry–Academia Gap

As discussed earlier in the report, recently there has been a significant progress in academic research in multi-agent path finding. Computer games are often mentioned as an application domain in research papers, and game maps have been used as benchmark data. However, it appears that there is a need for a better alignment between problems addressed in academic research and problems that are truly relevant to game developers. Recent academic research has focused on objectives such as computing optimal solutions, or scaling sub-optimal methods to hundreds or even thousands of mobile units. Such objectives are important on their own but, in many real-time strategy games, solution optimality is not strictly needed, and there can be relatively few units interacting on a portion of a map. In such games, the ability to encircle and attack a moving enemy unit, which combines elements of multi-agent collaborative pathfinding, multi-agent adversarial pathfinding, and multi-agent moving-target search, could be quite important.

We believe that a better industry–academia integration in multi-agent pathfinding could benefit if game developers make available precise definitions of some of their problems and interest, and possibly data to experiment with. They could be made available on their own, or be part of a competition open to academic researchers. Recently, Sturtevant has organized a Grid-Based Path Planning Competition[1]. This first edition focused on single-agent pathfinding on fully known, static maps. This could pave the way towards more diverse types of competition problems, such as multi-agent path planning.

In multi-agent pathfinding, non-interfering paths that bring each agent from its current state to its goal state must be planned. The agents are considered as friendly, and one agent may give the way to another one. For example, workers mining and transporting ore is such a problem. The complexity of a problem depends on the connectivity of the grid, the rules governing agent movements, the size of the grid, the number of obstacles, the number of agents. The challenge is to find solutions that avoid collisions and deadlocks in the context of fast computations. Future work consist in filling the gap between optimal algorithms solving relatively simple problems and tractable algorithms solving very complex problems but not optimally.

In adversarial multi-agent pathfinding, encircling behaviors, pursuit behaviors and/or escaping behaviors must be planned. For example, encircling an enemy unit to attack it, guarding one or several chokepoints, catching up a prey. Some agents are inimical and their meetings lead to fight whose outcomes can be the partial weakening or the complete destruction of some of the agents. However, some other agents are friendly and must coordinate their movements to reach a goal needing cooperation, such as circling a prey. The terminal conditions of the plan can be fuzzy and hard to determine, and planning can be continual. Future work include the integration of the adversarial multi-agent pathfinding solutions with game tree solutions to video games considered as two-player games.

3.2 From Smart Agents to Smart Maps

While single-agent path planning can naturally be modelled with an agent-centered approach, other problems may require a decision-making entity that is different from individual mobile units. For example, in multi-agent path planning, both views could be considered. A *distributed* approach would be agent-centered, allowing each agent to make decisions based on its (possibly incomplete) knowledge about the rest of the problem. Many multi-agent pathfinding algorithms, however, including most methods discussed in this report, assume the existence of a centralized decision maker.

Taking the idea further, we argue that part of the intelligence built in a path-finding module could be associated with parts of the environment in a distributed manner. For example, a notion of a *smart chokepoint* could be used to schedule incoming units through a narrow area of a map. The idea is similar to a traffic light system controlling an individual intersection on a road map. More generally, local portions of a map, not restricted to certain types of topologies could be responsible for a smooth navigation inside their own areas.

A dynamic environment offers an additional scope for developing intelligent map functionality. As pointed out earlier in the report, advanced pathfinding algorithms can use pre-computed datastructures for a faster path computation. Dynamic changes can invalidate underlying datastructures. A smart map should be able to know when to trigger a repair, what area of the map should be affected, how to perform a repair, and how to handle ongoing navigation while a local datastructure is still under repair.

[1] `http://movingai.com/GPPC/index.html`

3.3 Time and Memory Constraints

While common formulations of pathfinding problems, such as single-agent pathfinding on a static graph, have a low polynomial complexity, the challenge is to satisfy the ambitious CPU and memory limits set in actual games. There tends to be a speed–memory trade-off, exhibited, for example, by methods that perform pre-processing and use the cached results of pre-processing for a faster online pathfinding. Path quality is a third performance measure that must be considered. The challenge is to improve the performance on one or more such criteria, without paying a price on the remaining criteria.

Compressed path databases (CPDs) illustrate the speed–memory tradeoff well, being a fast but memory-consuming method. Compressed path databases are more memory intensive than other pathfinding methods, such as techniques that involve A* search. Thus, we use CPDs as a case study, discussing how its memory footprint could be improved.

Future efforts on reducing their size could aim at preserving path optimality, or could relax this requirement, as a tradeoff to a better size reduction. In games, the optimality of solutions is not seen as a must-have feature. Suboptimal paths that look reasonable to a user are acceptable. A promising idea that involves suboptimal paths is combining compressed path databases with hierarchical abstraction. Recently, Kring et al. [22] have extended the HPA* algorithm to cache (uncompressed) all-pairs shortest paths for small rectangular subparts of a map. The idea could be taken further to store such local all-pairs shortest paths in a compressed form, in the style of CPDs. Furthermore, at the global level, a compressed path database could be computed for the abstract graph that HPA* uses. This approach would result in a hierarchical, two-level compressed path database.

Finally, we advocate a closer interaction with communities working on related problems, such as robot navigation, journey planning on a road map, and multi-modal journey planning in a city. Similarities between such domains and pathfinding in games have encouraged, to some extent, the transfer of ideas across domains. Clearly, future common events, such as workshops at big conferences, would help tighten the connections.

4 Conclusion

Summarizing discussions held in the Pathfinding Workgroup at the Dagstuhl Seminar 12191, this chapter focused on recent advances, current limitations and possible future directions in pathfinding research. A comprehensive literature survey was beyond the purpose of our work. Instead, our survey was meant to set the grounds for identifying challenges and future work ideas.

It seems that research in pathfinding is not in a danger of fading. The recent first edition of the pathfinding competition suggests that the interest remains high even for the standard, well studied single-agent pathfinding on a fully known grid map. Yet, some diversification to the focus of future research might be desirable. For example, in problems such as multi-agent pathfinding, there seems to be a gap between academic research and commercial game development. The two communities could mutually benefit from each other if game developers make available descriptions of problem variants that are particularly relevant to a game, together with performance specs (speed, memory) expected from a solving algorithm.

References

1 Y. Björnsson, M. Enzenberger, R. Holte, and J. Schaeffer. Fringe Search: Beating A* at Pathfinding on Game Maps. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games CIG-05*, pages 125–132, 2005.

2 Yngvi Björnsson and Kári Halldórsson. Improved heuristics for optimal path-finding on game maps. In *Proceedings of the Conference on AI and Interactive Digital Entertainment AIIDE-06*, pages 9–14, 2006.

3 A. Botea, M. Müller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28, 2004.

4 Adi Botea. Ultra-fast Optimal Pathfinding without Runtime Search. In *Proceedings of the Conference on AI and Interactive Digital Entertainment (AIIDE-11)*, pages 122–127, 2011.

5 Vadim Bulitko, Yngvi Björnsson, and Ramon Lawrence. Case-based subgoaling in real-time heuristic search for video game pathfinding. *Journal of Artificial Intelligence Research (JAIR)*, 39:269–300, 2010.

6 Tristan Cazenave. Optimizations of data structures, heuristics and algorithms for path-finding on maps. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games CIG-06*, pages 27–33, 2006.

7 Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. In *Proc. of the 21st National Conference on Artificial intelligence*, AAAI-06, pages 942–947, 2006.

8 David Ferguson and Anthony (Tony) Stentz. Using Interpolation to Improve Path Planning: The Field D* Algorithm. *Journal of Field Robotics*, 23(2):79–101, February 2006.

9 A. V. Goldberg and C. Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms SODA-05*, pages 156–165, 2005.

10 M. Goldenberg, Ar. Felner, N. Sturtevant, and J. Schaeffer. Portal-based true-distance heuristics for path finding. In *Proceedings of the Symposium on Combinatorial Search SoCS-10*, pages 39–45, 2010.

11 Meir Goldenberg, Nathan Sturtevant, Ariel Felner, and Jonathan Schaeffer. The compressed differential heuristic. In *Proceedings of the National Conference on Artificial Intelligence AAAI-11*, pages 24–29, 2011.

12 Daniel Harabor and Adi Botea. Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games CIG-08*, pages 258–265, Perth, Australia, December 2008.

13 Daniel Harabor and Adi Botea. Breaking path symmetries in 4-connected grid maps. In *Proceedings of the Conference on AI and Interactive Digital Entertainment AIIDE-10*, pages 33–38, 2010.

14 Daniel Harabor, Adi Botea, and Phil Kilby. Path Symmetries in Undirected Uniform-cost Grids. In *In Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation (SARA-11)*, 2011.

15 Daniel Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In *Proceedings of the National Conference on Artificial Intelligence AAAI-11*, 2011.

16 P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Sciences and Cybernetics*, SSC-4(2):100–107, 1968.

17 R. Holte, M. Perez, R. Zimmer, and A. MacDonald. Hierarchical A*: Searching Abstraction Hierarchies Efficiently. Technical report, University of Ottawa, TR-95-18, 1995.

18 Marcelo Kallmann, Hanspeter Bieri, and Daniel Thalmann. Fully dynamic constrained delaunay triangulations. In G. Brunnett, B. Hamann, H. Mueller, and L. Linsen, editors, *Geometric Modelling for Scientific Visualization*, pages 241–257. Springer-Verlag, Heidelberg, Germany, first edition, 2003. ISBN 3-540-40116-4.

19 M.M. Khorshid, R.C. Holte, and N. Sturtevant. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search (SoCS)*, pages 76–83, 2011.

20 R. Korf. Real-Time Heuristic Search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

21 D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 241–250, 1984.

22 Alex Kring, Alex J. Champandard, and Nick Samarin. DHPA* and SHPA*: Efficient Hierarchical Pathfinding in Dynamic and Static Game Worlds. In *Proceedings of the AI and Interactive Digital Entertainment Conference AIIDE-10*, pages 39–44, 2010.

23 Ramon Lawrence and Vadim Bulitko. Database-driven real-time heuristic search in videogame pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games*, 5(3):227–241, 2013.

24 Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius. Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). *Dagstuhl Reports*, 2(5):43–70, 2012.

25 Ryan Luna and Kostas E. Bekris. An efficient and complete approach for cooperative path-finding. In *Proc. of the National Conference on Artificial Intelligence AAAI-11*, 2011.

26 Gabriele Röger and Malte Helmert. Non-optimal multi-agent pathfinding is solved (since 1984). In *Proceedings of the Symposium on Combinatorial Search SoCS-12*, 2012.

27 Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. Efficient query processing on spatial networks. In *ACM Workshop on Geographic Information Systems*, pages 200–209, 2005.

28 Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI-11*, pages 662–667, 2011.

29 D. Silver. Cooperative pathfinding. *AI Programming Wisdom*, 2006.

30 Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the National Conference on Artificial Intelligence AAAI-10*, 2010.

31 Trevor Scott Standley and Richard E. Korf. Complete algorithms for cooperative pathfinding problems. In *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI-11*, pages 668–673, 2011.

32 N. Sturtevant, A. Felner, M. Barer, J. Schaeffer, and N. Burch. Memory-based heuristics for explicit state spaces. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 609–614, 2009.

33 Nathan Sturtevant. Memory-efficient abstractions for pathfinding. *Proceedings of the International Conference on Artificial Intelligence and Interactive Digital Entertainment AIIDE-07*, pages 31–36, 2007.

34 Nathan Sturtevant and Michael Buro. Partial Pathfinding Using Map Abstraction and Refinement. In *Proceedings of the National Conference on Artificial Intelligence AAAI-05*, pages 47–52, 2005.

35 Nathan Sturtevant and Michael Buro. Improving collaborative pathfinding using map abstraction. In *Proceedings of the International Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 80–85, 2006.

36 W. van der Sterren. Path Look-Up Tables – Small is Beautiful. In S. Rabin, editor, *AI Game Programming Wisdom 2*, pages 115–129, 2003.

37 K.-H. C. Wang and A. Botea. Fast and Memory-Efficient Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 380–387, 2008.

38 K.-H. C. Wang and A. Botea. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research JAIR*, 42:55–90, 2011.

39 Peter Yap, Neil Burch, Robert C. Holte, and Jonathan Schaeffer. Block A*: Database-Driven Search with Applications in Any-Angle Path-Planning. In *Proceedings of the National Conference on Artificial Intelligence AAAI*, 2011.