

On Case Base Formation in Real-Time Heuristic Search

Vadim Bulitko and Chris Rayner

Department of Computing Science
 University of Alberta
 Edmonton, AB, T6G 2E8, Canada
 { bulitko | rayner } @cs.ualberta.ca

Ramon Lawrence

Computer Science
 University of British Columbia
 Kelowna, BC, V1V 1V7, Canada
 ramon.lawrence@ubc.ca

Abstract

Real-time heuristic search algorithms obey a constant limit on planning time per move. Agents using these algorithms can execute each move as it is computed, suggesting a strong potential for application to real-time video-game AI. Recently, a breakthrough in real-time heuristic search performance was achieved through the use of case-based reasoning. In this framework, the agent optimally solves a set of problems and stores their solutions in a case base. Then, given any new problem, it seeks a similar case in the case base and uses its solution as an aid to solve the problem at hand. A number of *ad hoc* approaches to the case base formation problem have been proposed and empirically shown to perform well. In this paper, we investigate a theoretically driven approach to solving the problem. We mathematically relate properties of a case base to the suboptimality of the solutions it produces and subsequently develop an algorithm that addresses these properties directly. An empirical evaluation shows our new algorithm outperforms the existing state of the art on contemporary video-game pathfinding benchmarks.

1 Introduction

Heuristic search is widely used in Artificial Intelligence, and plays a key role in many modern video games. Many search algorithms, such as the ubiquitous A^* , are uninterruptible until the complete solution is computed. In this paper we are concerned with *real-time* search, where the agent needs to produce solution fragments in rapid increments, often well before a complete solution can be computed. Real-time search interleaves planning and acting, keeping the planning time per action independent of the search space size.

Because the real-time agent acts without full knowledge of an optimal solution, it can commit to suboptimal actions. Since it acts in real-time, it is unable to correct them. As a result, the solutions produced when it does reach the goal can be longer than necessary and aesthetically unsatisfying due to unnecessary state re-visitation. Since the seminal work by Korf (1990), many researchers have attempted to improve the quality of solutions produced by real-time heuristic search (Furcy and Koenig 2001; Shue, Li, and Zamani 2001; Koenig 2004; Hernández and Meseguer 2005; Bulitko and Lee 2006; Koenig and Likhachev 2006; Rayner

et al. 2007; Bulitko et al. 2007; Björnsson, Bulitko, and Sturtevant 2009). Despite these and many other improvements, the resulting agents still exhibited state re-visitation and occasionally produced highly suboptimal solutions.

A leap in solution quality came with D LRTA* (Bulitko et al. 2008) which introduced case-based reasoning to real-time heuristic search. kNN LRTA* (Bulitko, Björnsson, and Lawrence 2010) and HCDPS (Lawrence and Bulitko 2010) made case-based reasoning more efficient. HCDPS has virtually eliminated state re-visitation, becoming the state of the art in the domain of video-game pathfinding.

The performance gains made by these algorithms are due to their use of a *case base* of pre-computed optimal solutions. When faced with a never-before-seen problem, the agent searches its case base for an *a priori* optimally solved problem or “case” similar to the one at hand. This case then constitutes a fixed fragment of the agent’s final solution – much as we use freeways when driving – freeing the agent to focus on the simpler task of connecting the fragment’s end-points to the start and goal. Consequently, the quality of the solution depends crucially on the quality of the case base.

In this paper we tackle the problem of building compact case bases which lead to high-quality solutions when used online. While existing work in the literature addresses *ad hoc* formulations of this problem, we formalize it as a discrete optimization problem for the first time. From this, we argue that computing an optimal case base or even a greedy approximation to it may be intractable. We then connect solution suboptimality to case base properties. One such property is “coverage”: the more problems covered by the case base, the more likely the agent is to have a partial solution for any problem it faces. While optimizing coverage for a fixed number of cases is NP-hard, it is connected to a simpler state coverage problem which allows an efficient greedy approximation. We take advantage of this fact to build a simple case base formation algorithm. We evaluate it empirically using a benchmark set of *Dragon Age: Origins* maps (Sturtevant 2012), showing performance gains over the existing state of the art (Lawrence and Bulitko 2010).

2 Terminology, Notation and Conventions

Heuristic search problems. A *search graph* is a directed graph (S, E) with a finite set of states S and a finite set of edges E . Each edge $(s_1, s_2) \in E$ has a positive cost

$c(s_1, s_2)$ associated with it. A *search problem* is defined by two states $s \in S$ and $g \in S$ designated as *start* and *goal*. The set of all solvable problems on a search graph (S, E) is denoted $P(S, E)$ or just P for short. We adopt the standard assumption of safe explorability (i.e., the goal state is reachable from any state reachable from the start state).

Real-time heuristic search. A *real-time heuristic search agent* is situated in the agent-centered search framework (Koenig 2001). Specifically, at every time step, a search agent has a single *current state* in the search graph which it can change by taking an action (i.e., traversing an edge out of the current state). Planning of the next action is followed by executing it. Real-time heuristic search imposes a constant time constraint on the planning time for every action, independently of the number of states $|S|$.

Given a search problem $(s, g) \in P$, the agent starts in s and solves the problem by arriving at the goal state g . The total sum of the edge weights traversed by an agent a between s and g is called the *solution cost* $c_a(s, g)$. If $c_a(s, g)$ is minimal, it is called the *optimal solution cost* and is denoted by $c^*(s, g)$. The search space comes with a heuristic $h(s, g)$ – an estimate of $c^*(s, g)$. The perfect heuristic is $h^* \equiv c^*$. The *suboptimality* of solving (s, g) with the agent a is the ratio of a 's solution cost to the optimal solution cost: $\sigma_a(s, g) = c_a(s, g)/c^*(s, g)$. We report suboptimality as percentage points over optimal: $100 \cdot (\sigma_a(s, g) - 1)\%$.

Hill-Climbability. Given a problem $(s, g) \in P$, the hill-climbing agent evaluates $h(s', g)$ for each neighbor s' of its current state, s_\diamond . It then moves to the neighboring state s' with the lowest $c(s_\diamond, s') + h(s', g)$. Hill-climbing fails when the minimum $h(s', g)$ is no smaller than $h(s, g)$. In this case we say that g is not hill-climbable from s .

If a goal g can be reached in m repeat iterations of this process beginning at a state s , we say that g is *hill-climbable* from s , and denote this as $s \rightsquigarrow_m g$. The resulting solution cost is denoted by $c_{\rightsquigarrow}(s, g)$,¹ and its suboptimality is denoted by $\sigma_{\rightsquigarrow}(s, g) = c_{\rightsquigarrow}(s, g)/c^*(s, g)$. Similarly, we say that the problem $(s', g') \in P$ is *bidirectionally hill-climbable* with $(s, g) \in P$ when $s \rightsquigarrow_m s' \ \& \ g' \rightsquigarrow_m g$. We denote this as $(s, g) \leftrightarrow_m (s', g')$.

3 Related Research

The use of pre-computed data structures is a common and effective approach in planning and heuristic search. The work in this paper belongs to a sub-category of these methods which uses the solutions to problems (or “cases”) computed *offline* to assist in solving any other problem *online* in real time. By doing so, these methods avoid many of the online storage costs of standard search methods, such as open and closed lists, learned heuristic values, etc.

This approach is commonly known as case-based reasoning (CBR) (Aamodt and Plaza 1994). CBR has been applied to planning problems (de Mántaras et al. 2005) as well as used to avoid heuristic search (Coello and dos Santos 1999) and for strategy planning in video games (Ontañón et al.

¹We omit m from $c_{\rightsquigarrow}(s, g)$ because it does not depend on m as long as $s \rightsquigarrow_m g$.

2007). The issues of case base construction, curation, and retention have also been investigated (de Mántaras et al. 2005; Smyth and McKenna 2001). However, there have been limited applications of CBR to pathfinding specific to video games. The challenges here are the fast storing and retrieving of easily compressed case paths.

The recently proposed Compressed Path Database (CPD) algorithm (Botea 2011) stores the optimal first action for every problem using an efficient compression routine. This enables fast real-time access to any optimal path through the state space, giving full problem coverage. However, it has an expensive pre-computation phase and can produce large databases, with published data suggesting a memory-optimized CPD requires $|P| \cdot 0.08$ bits to store. As a result, a CPD can grow to 184 megabytes on modern maps, with no clear way to trim it to a desired size.

In video games, and in particular video-game pathfinding, optimal solutions can be less important than a game developer’s control over the algorithm’s memory footprint. The first real-time heuristic search algorithm with user-controlled database size was D LRTA* (Bulitko et al. 2008), which stored partial solutions to problems between representatives of partitions of the state space. These pre-computed solutions were then used to guide the classic LRTA* (Korf 1990) when solving new problems online. The partitions were procedurally generated using clique abstraction which allows the user to coarsely control the resulting database size via adjusting the abstraction level. On the downside, the partitions were expensive to compute and topologically complex, leading to extensive state re-visitation by LRTA*.

Both issues were addressed by kNN LRTA* (Bulitko, Björnsson, and Lawrence 2010) which solved a user-specified number of random problems offline and stored their solutions in a case base. Faced with a new problem online, it retrieved a similar case and used it to guide LRTA*. While kNN LRTA* gave fine-grained control over the memory footprint, its random case selection gave no guarantee that every problem had a relevant case in the case base. When a relevant case could not be found, the LRTA* agent would run unassisted, often falling into heuristic depressions and ending up with excessive state re-visitation.

The current state of the art, HCDPS (Lawrence and Bulitko 2010), guarantees that every problem will have a relevant case by partitioning the search space. Unlike D LRTA*, however, the partitions are built to make it easy for a hill-climbing agent to reach a partition’s representative from any state in the partition. HCDPS then pre-computes (near) optimal paths between representatives of any two partitions and stores them in its case base. Online, the shape of the partitions affords the use of a simpler hill-climbing agent instead of LRTA*, thereby, for the first-time in real-time heuristic search, (virtually) eliminating state re-visitation and maintaining a zero online memory footprint. The case base size is coarsely controllable via the hill-climbing cutoff m .

These algorithms take *ad hoc* approaches to forming a case base and, subsequently, trading the size of the case base for solution suboptimality. The first contribution of this paper is a theoretically grounded framing of the problem.

4 Case-Based Reasoning

In this section we frame subgoal-assisted real-time heuristic search as case-based reasoning. A *case base* $R \subseteq P(S, E)$ is a set of problems stored with optimal solutions pre-computed offline. When faced with any new problem (s, g) online, the case-based agent can use the pre-computed solution to any $(s', g') \in R$ as a part of its solution to (s, g) . Intuitively, a relevant case is one for which s and s' are close, and g and g' are close. Hence, we follow the spirit of kNN LRTA* and HCDPS and say that a case/problem (s', g') is *relevant* to problem (s, g) if (s', g') is bidirectionally hill-climbable with (s, g) : $(s, g) \rightsquigarrow_m (s', g')$.

Given a relevant case (s', g') with which to solve (s, g) , our agent hill-climbs from s to s' , follows the pre-computed solution to (s', g') , then hill-climbs from g' to g . The resulting solution suboptimality $\sigma_{(s', g')}(s, g)$ depends on how (s', g') relates to (s, g) . In the event that the case base does not contain a relevant case, the agent invokes a complete real-time search algorithm α° , such as LRTA* (Korf 1990).

In this context we say a problem $(s, g) \in P$ is *covered* by the case base R under cutoff m if there exists a case $(s', g') \in R$ such that $(s, g) \rightsquigarrow_m (s', g')$. Accordingly, given a case $r \in R$, its *coverage* of P is the set of all problems from P which are covered by r . Formally, r 's coverage is denoted $\mathcal{C}_m(r, P) = \{p \in P \mid p \rightsquigarrow_m r\}$. The coverage of the entire case base R is the union of the coverage of its individual cases: $\mathcal{C}_m(R, P) = \bigcup_{r \in R} \mathcal{C}_m(r, P)$.

Case Base Formation Problem. Given the case-based reasoning framework, the critical factor in search performance (i.e., solution suboptimality, case base size, and case base generation time) is the method of case base formation. For the rest of the paper we consider the case base formation problem, formulating it precisely in this section.

We focus on solution suboptimality and thus attempt to solve the following discrete optimization problem: to form a case base $R \subseteq P$ of size n that minimizes *the expected suboptimality* of solving any problem in P with R . Formally:

$$\begin{aligned} & \underset{R \subseteq P}{\text{minimize}} && \sigma_R(P) && (1) \\ & \text{subject to} && |R| = n \end{aligned}$$

This objective presents a challenging optimization scenario. A brute-force algorithm would require evaluating $\sigma_R(P)$ $\binom{|P|}{n}$ times which is clearly intractable for any non-trivial values of $|P|$ and n . The objective (1) also does not show any exploitable structure, since $\sigma_R(P)$ is neither submodular nor monotone – properties which would afford good approximations. Even by resorting to a greedy algorithm, starting with $R = \emptyset$ and adding cases one by one according to the rule,

$$R \leftarrow R \cup \left\{ \arg \min_{r \in P} \sigma_{R \cup \{r\}}(P) \right\}, \quad (2)$$

we face evaluating $\sigma_{R \cup \{r\}}(P)$ a full $|P|$ times per iteration. Each such evaluation also requires solving all problems in P (often with the default algorithm). With $|P|^2 = |S|^4$ problems being solved per iteration, even this is intractable.

In this paper, we attack this optimization problem via a theoretically motivated angle. At the core of this approach is a mathematical analysis of $\sigma_R(P)$, not considered by past work. We then identify key principles for reducing the expected solution suboptimality of a case base which eventually leads to new computationally efficient algorithms.

5 Suboptimality Analysis

In this section we introduce machinery with which to directly analyze the expected suboptimality of a case base. We first represent solution suboptimality when solving problem $(s, g) \in P$ with a relevant case $(s', g') \in R \subseteq P$ as:

$$\sigma_{(s', g')}(s, g) = \frac{c_{\rightsquigarrow}(s, s') + c^*(s', g') + c_{\rightsquigarrow}(g', g)}{c^*(s, g)}. \quad (3)$$

If one or more cases in R are relevant to (s, g) then the case yielding the lowest solution cost is assumed to be used. When no relevant case is present in the case base then a default algorithm α° (e.g., LRTA*) is used. Thus, the suboptimality of solving a problem (s, g) with a case base R is:

$$\sigma_R(s, g) = \begin{cases} \min_{(s', g') \in R, (s, g) \rightsquigarrow_m (s', g')} \sigma_{(s', g')}(s, g), & \text{if } (s, g) \in \mathcal{C}_m(R, P) \\ \sigma_{\alpha^\circ}(s, g), & \text{otherwise.} \end{cases}$$

Given a case base R , the expected suboptimality of solving any problem $(s, g) \in P$ with R , denoted $\sigma_R(P)$, is the expected value of $\sigma_R(s, g)$ when a problem (s, g) is drawn uniformly randomly from P :

$$\begin{aligned} \sigma_R(P) &= E_{(s, g) \in P} [\sigma_R(s, g)] \\ &= E_{(s, g) \in P} [P_\exists \cdot \min \sigma_{(s', g')}(s, g) + (1 - P_\exists) \cdot \sigma_{\alpha^\circ}(s, g)] \end{aligned}$$

where P_\exists is the probability that a problem drawn randomly from P has a relevant case in R : $(s, g) \in \mathcal{C}_m(R, P)$.

Key insights. The analysis above gives us three insights into case base formation, as follows.

The first is *coverage*: if we expect that solving a problem with a case will give a better solution than the default algorithm produces (i.e., $\sigma_{\alpha^\circ}(s, g) \gg \sigma_{(s', g')}(s, g)$), then by increasing P_\exists we can decrease $\sigma_R(P)$.

The second is *HC overhead*: from the numerator in (3), it is clear that having cases closer to the problem at hand will decrease the hill-climbing overhead $c_{\rightsquigarrow}(s, s') + c_{\rightsquigarrow}(g', g)$.

The third is *curation*: when we lack a case for (s, g) , $\sigma_{\alpha^\circ}(s, g)$ should be low. We should prefer case(s) for problems on which the default algorithm is highly suboptimal.

In the following we look at optimizing coverage and HC overhead, leaving case base curation for future work.

6 Optimizing Coverage

In this section, we use the first insight (coverage) to develop a simple optimization approach to generating case bases. We call our approach Greedy State Cover (GSC). A subsequent empirical evaluation will reveal GSC to be capable of building case bases with better solution suboptimality than HCDPS, with the added benefit of flexible case base size.

To improve *problem* coverage, we should form a case base to increase the chances of having a case relevant to a random benchmark problem. Stated as an optimization problem:

$$\begin{aligned} & \underset{R \in 2^P}{\text{maximize}} && |C_m(R, P)| \\ & \text{subject to} && |R| = n. \end{aligned} \quad (4)$$

This objective function is non-negative, monotone and submodular and thus allows for an efficient greedy approximation. However, computing only the first iteration of such a greedy algorithm:

$$R \leftarrow R \cup \left\{ \arg \max_{r \in P} |C_m(R \cup \{r\}, P)| \right\}$$

calculates $C_m |P|$ times which can be intractable.

We can reduce the complexity by factoring the set of problems P as $S \times S$ and looking to cover S instead. Specifically, the *state-coverage* of $t \in S$ is the set of states $\mathcal{B}_m(t, S) = \{s \in S \mid s \rightsquigarrow_m t \ \& \ t \rightsquigarrow_m s\}$. The coverage of a set of states T is the union of the coverage of its individual states: $\mathcal{B}_m(T, S) = \bigcup_{t \in T} \mathcal{B}_m(t, S)$. An analogous (but not equivalent) optimization problem is then:

$$\begin{aligned} & \underset{T \in 2^S}{\text{maximize}} && |\mathcal{B}_m(T, S)| \\ & \text{subject to} && |T| = \sqrt{n}. \end{aligned} \quad (5)$$

Note that while going from 2^P to 2^S represents a significant combinatorial reduction, the problem is still challenging.

Theorem 1 The optimization (5) is NP-hard.

Proof. For the sake of brevity, we point out the semantic equivalence of *node domination* of node v and $\mathcal{B}_1(v, S)$. \square

Greedy State Cover (GSC). The objective function is non-negative, monotone and submodular and thus allows for an efficient greedy approximation by starting with $T = \emptyset$ and incrementally growing it as $T \leftarrow T \cup \{\arg \max_{t \in S} |\mathcal{B}_m(T \cup \{t\}, S)|\}$. We implement this rule in Algorithm 1 and show that doing so brings us within 63% of optimal state coverage when $|T|$ is fixed:

Theorem 2 Algorithm 1 produces T such that $|\mathcal{B}_m(T, S)| \geq (1 - e^{-1})|\mathcal{B}_m(T^*, S)|$ where T^* is an optimal solution to (5).

Proof. Follows from (Nemhauser, Wolsey, and Fisher 1978).

Algorithm 1 Greedy state-cover (GSC) trailhead selection

```

1:  $T \leftarrow \emptyset$ 
2:  $U \leftarrow S$ 
3: while  $U \neq \emptyset$  &  $|T| < \sqrt{n}$  do
4:    $r \leftarrow \arg \max_{t \in S} |\mathcal{B}_m(t, U)|$ 
5:    $T \leftarrow T \cup \{r\}$ 
6:    $U \leftarrow U \setminus \mathcal{B}_m(t, U)$ 
7: end while
8: return  $T$ 

```

Algorithm 1 starts with $T = \emptyset$ (line 1) and the set U of states yet to cover being the set of all states S (line 2). While there are still states to cover and we have not reached the

target size of \sqrt{n} states (line 3), we select the state t with the highest state coverage (line 4),² add it to the set T (line 5), and update the set of states still to cover (line 6).

Note that calculating $|\mathcal{B}_m(t, U)|$ in line 4 is local to state t as only neighboring states up to m moves away need to be considered. Consequently, the overall complexity of line 4 is $O(|S|)$ which is a substantial reduction over the $\Omega(|S|^4)$ complexity of the original greedy selection rule (2).

The output of Algorithm 1 is a set T of states fully or partially covering S . We call these states *trailheads* and form our final GSC case base as the cross-product $R = T \times T$.

6.1 An Illustration

GSC is similar to HCDPS (Lawrence and Bulitko 2010) inasmuch as we partition the state space into regions whose states are bidirectionally hill-climbable with a single representative state, or trailhead. Each trailhead covers its region in the sense of \mathcal{B}_m . But HCDPS does not explicitly perform an optimization when selecting trailheads. Instead, it selects trailheads either randomly from U or in a domain-specific systematic way. Thus, for a fixed \sqrt{n} , HCDPS trailhead sets tend to have less coverage than GSC trailhead sets. HCDPS also tends to need more trailheads to achieve full coverage.

We demonstrate this effect on a gridworld map (lak202d) from the video game *Dragon Age: Origins*. The trailhead set T_{GSC} is generated with GSC. T_{HCDPS} is generated by HCDPS (i.e., selecting each trailhead randomly from the yet uncovered area of S). Both trailhead sets fully cover the state space, $\mathcal{B}_m(T_{\text{GSC}}, S) = \mathcal{B}_m(T_{\text{HCDPS}}, S) = S$, but more HCDPS trailheads are needed to achieve full coverage: 170 versus 139 trailheads. This is because the per-trailhead coverage is not explicitly maximized by HCDPS. On the upside, for a given state $s \in S$, the trailhead covering s tends to be closer with T_{HCDPS} than with T_{GSC} .

We can now generate the case base R_{GSC} as $T_{\text{GSC}} \times T_{\text{GSC}}$ and the case base R_{HCDPS} as $T_{\text{HCDPS}} \times T_{\text{HCDPS}}$. Since T_{HCDPS} has more trailheads than T_{GSC} , the case base R_{HCDPS} is larger than R_{GSC} : 0.0742% versus 0.0496% of all problems. Both cover the problem space, $C_m(R_{\text{GSC}}, P) = C_m(R_{\text{HCDPS}}, P) = P$, but the larger R_{HCDPS} gives denser coverage of P . As a result, for a given problem $(s, g) \in P$, the hill-climbing overhead $c_{\rightsquigarrow}(s, s') + c_{\rightsquigarrow}(g', g)$ between (s, g) and the closest relevant case $(s', g') \in R$ is expected to be lower with R_{HCDPS} than with R_{GSC} . Consequently, the suboptimality with the HCDPS case base is better than with GSC: 7.94% versus 9.31%, as shown in Figure 1.

6.2 Reducing HC Overhead

GSC's hill-climbing overhead can be reduced by lowering the cutoff limit m when computing \mathcal{B}_m in lines 4 and 6 of Algorithm 1. Note that the lowered m is used only when generating the trailhead set; it does not change the definition of a relevant case when solving problems online.

Temporarily lowering m during trailhead formation results in a denser placement of the trailheads and a larger trailhead set (179 versus 139 trailheads). Figure 1 compares the resulting case base sizes and solution suboptimality with

²Ties are broken randomly.

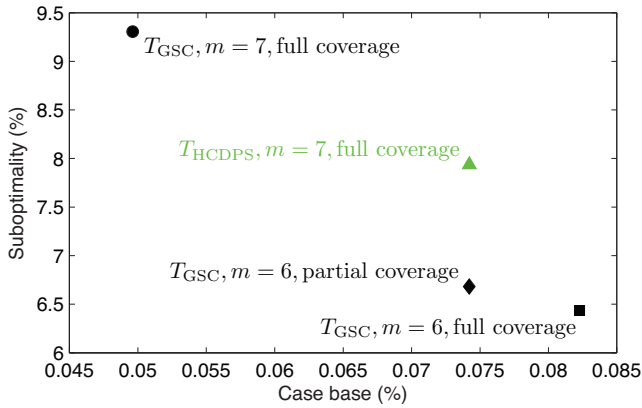


Figure 1: GSC case bases versus HCDPS case bases.

$R_{\text{HCDPS}}, m = 7$ and $R_{\text{GSC}}, m = 6$. The new GSC case base is larger than that of HCDPS (0.0823% versus 0.0742% of all problems) but has lower suboptimality (6.44% versus 7.94%) due to lower hill-climbing overhead.

6.3 Optimizing Coverage and HC Overhead

So far we have either improved case base size (by way of GSC instead of HCDPS) or improved solution quality (by running GSC with a temporarily lowered cutoff). In this section we improve size and quality together.

To do so, we run GSC with a temporarily lowered cutoff but set a restrictive limit on the number of trailheads. As a result, the while loop in Algorithm 1 will exit as soon as $|T|$ reaches \sqrt{n} , before all states are covered ($U = \emptyset$). In our running example, $T_{\text{HCDPS}}, m = 7$ achieves full coverage with 170 trailheads while $T_{\text{GSC}}, m = 6$ requires 179 trailheads for full coverage. By keeping $m = 6$ but setting $\sqrt{n} = 170$ we end up with a trailhead set of the same size as $T_{\text{HCDPS}}, m = 7$. While the resulting coverage of T_{GSC} will be partial with $m = 6$, the resulting coverage with $m = 7$ (used when finding relevant cases online) is likely to be full or close to full.³ Thus, for most problems the hill-climbing overhead will be reduced due to the denser trailhead placement. A few problems will end up without a relevant case, leading to the invocation of the default algorithm (e.g., LRTA*). In practice, the former effect is stronger and the overall solution quality will be better than with an HCDPS case base of equal size. In our example (Figure 1) the suboptimality is improved: 6.68% versus 7.94%.

7 Empirical Evaluation

We now present an empirical test of the hypothesis that our optimization-inspired GSC algorithm can consistently produce case bases similar in size but lower in suboptimality than HCDPS. We use the *de facto* standard testbed of video-game pathfinding. Similar testbeds have been used to evaluate D LRTA*, kNN LRTA* and HCDPS.

We use ten binary grid maps taken from the commercial video game, *Dragon Age: Origins*. The maps come from a recent standard pathfinding benchmark repository (Sturtevant 2012) and are shown with their statistics (dimensions,

³It happens to be full in our example.

obstacle density, number of states and problems) in Figure 2. Each open grid cell on the map is a state, and up to four edges connect each state to its neighbors — the four grid cells in the cardinal directions — all with a weight of 1. Manhattan distance is used as the heuristic.

The maps have between 3.1 and 8.9 thousand states and 9.6 and 79.1 million problems. These small maps enable a large scale study of database effects. Since both algorithms are randomized, we generate 500 case bases for each of the HCDPS parameters, and 1000 for each of the GSC parameters. A problem set containing 10 thousand problems is used to evaluate both algorithms. For each problem, the suboptimality of the case resulting in the lowest possible suboptimality is recorded, but if the problem is not covered by the case base then the suboptimality of LRTA* is recorded. We report average suboptimality (as a percentage) and case base size (as a percentage of all problems).

All experiments were run on a six-core Intel Core i7-980X CPU. Data structures and MATLAB/C code were reused when possible to make timings as comparable as possible.

Results. For each map we ran HCDPS with m specified⁴ in the m_H column in Table 1, and used the resulting case base to solve the problem set. This is repeated 500 times for each map as HCDPS is a randomized algorithm. The range of case base sizes (as a percentage of the number of all problems on the map⁵) is denoted “ $|R|$ range”. The solution suboptimality (%) averaged over the 10 thousand problems and 500 randomized case bases is listed as σ_H .

Next we ran GSC with m listed in column m_G . The number of trailheads was set to \sqrt{n} with n sampled randomly from the $|R|$ range, and the cross-product of the trailhead set was used as a case base to solve the problem set. Solution suboptimality over the same 10 thousand problems and across 1000 trials is listed as σ_G . The percent solution suboptimality improvement over HCDPS is listed as Δ , and is calculated as $100 \cdot ((\sigma_H - \sigma_G) / \sigma_H - 1)\%$.

In support of our hypothesis, GSC produces case bases of the same size but with the mean improvement of 7% in solution suboptimality over HCDPS. The price paid offline is up to an extra second of case base computation.

Map	$ R $ range	m_H	σ_H	m_G	σ_G	Δ
brc300d	0.06–0.09	7	6.3	6	5.9	6.2
brc502d	0.05–0.07	7	9.7	5	8.3	15.1
den001d	0.03–0.05	8	8.3	6	7.6	9.4
den020d	0.09–0.12	7	11.4	6	9.9	13.0
den206d	0.07–0.09	7	9.1	6	8.7	4.4
den308d	0.06–0.09	7	12.8	6	12.5	2.7
den901d	0.01–0.02	10	12.1	8	11.3	6.7
hrt001d	0.02–0.04	10	17.6	9	17.2	2.7
lak202d	0.07–0.09	7	7.1	6	6.5	8.8
orz302d	0.06–0.08	7	9.3	6	9.1	1.9
Mean	0.05–0.07	7.7	10.4	6.4	9.7	7.1

Table 1: Experimental results. HCDPS and GSC suboptimality are denoted σ_H and σ_G respectively.

⁴ m is picked by trial and error and is related to the size and geometry of the map.

⁵Our case bases ranged from 100 to 500 kilobytes per map.



Figure 2: The ten Dragon Age: Origins maps used in our experiments.

8 Conclusions and Future Work

The introduction of case-based reasoning to real-time heuristic search has led to dramatic performance improvements. However, case bases for this domain have so far been built in an *ad hoc* or domain-specific fashion. We have framed the problem more generally as a discrete optimization problem, reduced it to a state coverage problem, and suggested an efficient approximation algorithm. Our empirical evaluation indicates our approach outperforms the state of the art while allowing flexible control over case base size.

A promising direction for future work follows the third insight from Section 5 — case base curation. An efficient predictor of LRTA* suboptimality could allow for reducing case base size while retaining only the most valuable cases.

9 Acknowledgements

This research was supported by NSERC and iCORE. We are grateful to our anonymous reviewers for valuable feedback.

References

- Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–59.
- Björnsson, Y.; Bulitko, V.; and Sturtevant, N. 2009. TBA*: Time-bounded A*. In *IJCAI*, 431 – 436.
- Botea, A. 2011. Ultra-fast optimal pathfinding without run-time search. In *AIIDE*, 122–127.
- Bulitko, V., and Lee, G. 2006. Learning in real time search: A unifying framework. *JAIR* 25:119–157.
- Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *JAIR* 30:51–100.
- Bulitko, V.; Luštrek, M.; Schaeffer, J.; Björnsson, Y.; and Sigmundarson, S. 2008. Dynamic control in real-time heuristic search. *JAIR* 32:419 – 452.
- Bulitko, V.; Björnsson, Y.; and Lawrence, R. 2010. Case-based subgoalting in real-time heuristic search for video game pathfinding. *JAIR* 39:269–300.
- Coello, J. M. A., and dos Santos, R. C. 1999. Integrating cbr and heuristic search for learning and reusing solutions in real-time task scheduling. In *ICCB*, 89–103.
- de Mántaras, R. L.; McSherry, D.; Bridge, D. G.; Leake, D. B.; Smyth, B.; Craw, S.; Faltings, B.; Maher, M. L.; Cox, M. T.; Forbus, K. D.; Keane, M. T.; Aamodt, A.; and Watson, I. D. 2005. Retrieval, reuse, revision and retention in case-based reasoning. *Knowl. Eng. Review* 20(3):215–240.
- Furcy, D., and Koenig, S. 2001. Combining two fast-learning real-time search algorithms yields even faster learning. In *Proceedings of the European Conference on Planning*, 391–396.
- Hernández, C., and Meseguer, P. 2005. Improving convergence of LRTA*(k). In *IJCAI workshop*, 69–75.
- Koenig, S., and Likhachev, M. 2006. Real-time adaptive A*. In *AAMAS*, 281–288.
- Koenig, S. 2001. Agent-centered search. *AI Magazine* 22(4):109–132.
- Koenig, S. 2004. A comparison of fast search methods for real-time situated agents. In *AAMAS*, 864–871.
- Korf, R. 1990. Real-time heuristic search. *AIJ* 42(2–3):189–211.
- Lawrence, R., and Bulitko, V. 2010. Taking learning out of real-time heuristic search for video-game pathfinding. In *Australasian Joint Conference on AI*, 405–414.
- Nemhauser, G.; Wolsey, L.; and Fisher, M. 1978. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming* 14(1):265–294.
- Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2007. Case-based planning and execution for real-time strategy games. In *ICCB*, 164–178.
- Rayner, D. C.; Davison, K.; Bulitko, V.; Anderson, K.; and Lu, J. 2007. Real-time heuristic search with a priority queue. In *IJCAI*, 2372–2377.
- Shue, L.-Y.; Li, S.-T.; and Zamani, R. 2001. An intelligent heuristic algorithm for project scheduling problems. In *Proc. of 32nd Annual Meeting of the Decision Sciences Institute*.
- Smyth, B., and McKenna, E. 2001. Competence models and the maintenance problem. *Computational Intelligence* 17(2):235–249.
- Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *IEEE TCIAIG* 4(2):144 – 148.