Dynamic Potential Search on Weighted Graphs

Daniel Gilon
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
(daniel.gilon@gmail.com)

Ariel Felner
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
(felner@bgu.ac.il)

Roni Stern
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
(roni.stern@gmail.com)

Abstract

Dynamic Potential Search (DPS) is a recently introduced search algorithm that returns a *bounded-suboptimal cost* solution. DPS orders nodes in the open-list based on their *potential* which is a combination of both the g- and h-values of a node. In this paper we study the behavior of DPS on weighted graphs. In particular, we develop a new variant of DPS, called DPSU which calculates the potential by counting one for each edge regardless of its costs. We develop an eager version and a restrained version of DPSU. We then compare all these algorithms on a number of weighted graphs and study the pros and cons of each of them.

1 Introduction and Overview

A* is a *best-first search* algorithm which orders nodes in OPEN according to $f(n) = g(n) + h(n)$. If $h(n)$ is an *admissible* heuristic (i.e., is always a lower bound on cost of the remaining path to goal) then A* is guaranteed to find an optimal (lowest-cost) solution. However, in many cases, (e.g., video games, embedded systems or mobile apps) one must settle for a suboptimal solution by trading time/memory for solution quality. Two such non-optimal search settings are:

(1) Bounded cost search (Stern et al. 2014) (denoted here as BCS). In BCS(C) we are given a constant C and the task is to find a solution with cost $\leq C$.

(2) Bounded suboptimal search (denoted here as BSS). In BSS(B) we are given a bound B and the task is to find a solution with cost $\leq B \times P_{opt}$ where P_{opt} is the cost of the optimal solution.

Potential Search (PS) (Stern et al. 2014) is an algorithm specifically designed for BCS. PS is a best-first search that chooses to expand the node n from OPEN with the largest "potential", which is defined as $u(n) = \frac{C-g(n)}{h(n)}$.[1] In addition, for an admissible h, the algorithm prunes any node n for which $f(n) = g(n) + h(n) > C$, as it will not lead to a solution within the bound. Intuitively, given a node n, $C - g(n)$ is an upper bound on any path in the subtree below node n that will still be within the bound C. Dividing this by the estimated cost to the goal, $h(n)$, gives its *potential*, i.e., how likely there is a goal node within the bound in the subtree rooted at n. Stern et al. (2014) showed that under certain conditions PS expands in every iteration the node that is most likely to be part of a solution that is within the bound. PS will find a desired solution if one exists. PS was shown to be very effective in finding BCS solutions.

Gilon, Felner and Stern (2016) showed that under some conditions, one can migrate algorithms from BCS to BSS and vice versa. In particular, they migrated PS to the BSS setting and introduced a new BSS algorithm called *dynamic potential search* (DPS). In PS the cost bound C remains constant throughout the search. DPS dynamically modifies C and sets $C = f_{min} \times B$, where f_{min} is the minimal f-value in OPEN. Then, DPS expands the node in OPEN with the largest potential with respect to the current C. Therefore, DPS is guaranteed to find a solution within the desired suboptimality bound. Gilon et al. (2016) experimentally compared DPS to other known algorithms that are designed for BSS, namely to WA* (Pohl 1970) and to EES (Thayer and Ruml 2011). They showed that (with some exceptions) in unit-edge cost domains DPS significantly outperforms EES and WA* by a factor of up to 180 in the number of nodes expanded and in CPU time. But, DPS was much less effective than EES on a number of domains with non-unit edge costs such as the *inverse 15 puzzle* (see below).

In this paper we further investigate DPS under weighted graphs. We introduce *Dynamic Potential Search with Unit Estimations* (DPSU) which uses the potential formula but on an abstract problem which uses the number of edges and not their costs. We develop an eager version and a restrained version of DPSU. We theoretically and experimentally compare the two versions of DPSU to DPS and to EES on a number of benchmark domains and study their pros and cons. Restricted DPSU was found to provide the best performance and is the most robust across all the domains we studied.

2 Focal Search

A large number of BSS algorithms have been proposed over the years (see a nice survey by Thayer and Ruml, 2011). Most of these algorithms can be viewed as implementations of a general search framework that is sometimes called *Focal Search* (Pearl and Kim 1982; Ebendt and Drechsler 2009; Valenzano et al. 2013). Focal Search is a type of best-first

[1] For cases where $h(n) = 0$, $u(n)$ is defined to be $+\infty$ if $C \geq g(n)$, causing such nodes to be expanded first, and $-\infty$ if $C < g(n)$, ignoring such nodes.

Algorithm 1: Focal Search: main procedure

```
1  focal-search(start state S)
2      OPEN ← {S};
3      FOCAL ← {S};
4      while FOCAL ≠ ∅ do
5          best ← ChooseNode(FOCAL)
6          Remove best from FOCAL and OPEN
7          if best is a goal then return best;
8          if f_min increased then FixFocal() ;
9          for n ∈ neighbors(best) do
10             Add n to OPEN
11             if f(n) ≤ B × f_min then add n to FOCAL ;
12         end
13     end
14 end
```

search that maintains two sets of nodes. Similar to any best-first search, it maintains OPEN which includes all nodes that were generated but not expanded. In addition, Focal Search maintains a list of nodes FOCAL $\subseteq$ OPEN defined as:

$$\text{FOCAL} = \{n \in \text{OPEN} \mid f(n) \leq B \times f_{min}\}$$

As a special case of best-first search, a Focal Search algorithm chooses in every expansion cycle a single node from OPEN and expands it, but it is constrained to only choose a node that is also in FOCAL. As the search progresses, f_{min} may increase. When this occurs, the range $B \times f_{min}$ also grows and more nodes from OPEN are added to FOCAL. This allows a broader range of nodes to be considered for expansion in subsequent expansion cycles. Once a goal node is chosen for expansion, the search halts. When h is admissible any Focal Search is a BSS algorithm. This is because f_{min} is a lower bound on P_{opt} and when a goal node t is expanded it must be in FOCAL, and thus $f(t) = g(t) \leq B \times f_{min} \leq B \times P_{opt}$. Algorithm 1 presents the main structure of Focal Search. Focal Search algorithms differ in which node from FOCAL they choose to expand (line 5).

Perhaps the simplest form of Focal Search is to choose the node in FOCAL with the smallest h-value. This algorithm is often referred to as A^*_ϵ (Ebendt and Drechsler 2009) although the original A^*_ϵ (Pearl and Kim 1982) is more general. In its basic form, Focal Search keeps FOCAL as a separate list, and whenever f_{min} increases, the relevant nodes from OPEN are identified and are added to FOCAL. This is done by `FixFocal` (line 8 in Algorithm 1) which incurs some computation overhead. However, in some cases it is possible to implement Focal Search without maintaining a separate list of nodes for FOCAL. For example, some decision rules for choosing which node to expand from OPEN guarantee that the node n chosen for expansion is in FOCAL, i.e., that $f(n) \leq B \times f_{min}$ (Valenzano et al. 2013).

3 DPS

DPS (Gilon, Felner, and Stern 2016) is a Focal Search BSS algorithm. Inherited from PS, DPS chooses the node n with the highest "potential", but now the potential relates to the likelihood of finding a goal below n with cost $\leq B \times f_{min}$. In other words, given the current FOCAL, we use the same `ChooseNode()` as PS but set $C = B \times f_{min}$. Formally, DPS chooses to expand a node n that maximizes:

$$ud(n) = \frac{B \times f_{min} - g(n)}{h(n)}$$

Gilon et al. (2016) showed that both Weighted A* (Pohl 1970; 1973) as well as DPS are focal searches in the sense that the nodes they choose to expand from OPEN according to their priority function is always in FOCAL. Thus, DPS and WA* need not store a separate FOCAL list. In their experiments Gilon et al. (2016) observed that DPS is usually very strong in unit edge cost domains but it was weak in some weighted edge-cost domains. For example, DPS was strong for the *heavy 15-puzzle*, where moving tile $\#X$ costs X, but it was very weak on the *inverse 15-puzzle*, in which moving tile $\#X$ costs $1/X$. This paper was sparked by our motivation to fix DPS to also work well for the *inverse 15-puzzle*. For this, we introduce the following variant of DPS.

4 DPS with Unit Estimations

DPS with Unit Estimations (DPSU) is a new variant of DPS. DPSU is motivated by the fact that it is usually easier to search when using unit costs than when using weighted costs (Wilt and Ruml 2011; 2014; Thayer, Benton, and Helmert 2012). Given a weighted graph, we define a *unit edge cost function* as an abstract cost function that only counts the number of edges. We thus use the following terms. $g^u(n)$ is a function that counts the number of edges from the start state to n. This is often referred to as the *depth* of n. Similarly, $h^u(n)$ is an admissible (lower bound) estimation on the number of edges from n to the nearest goal.[2] Clearly, $f^u(n) = g^u(n) + h^u(n)$ is a lower bound estimation on $OPT^u(n)$ which is the path with the smallest number of edges from the start to the (closest) goal via node n. DPSU uses the same potential formula as DPS but in its internal terms it uses g^u, h^u, and f^u_{min}, instead of g, h, and f_{min}. Formally, DPSU chooses to expand the node from OPEN that has the maximal *unit potential* defined as:

$$ud^u(n) = \frac{B \times f^u_{min} - g^u(n)}{h^u(n)}$$

Let G be a weighted graph and let G^u be an abstraction of G where all edges are of unit weights. On G^u, DPS and DPSU are equivalent. Running DPSU on G simulates the execution of DPS on G^u but with the following important difference. Once the first goal is chosen for expansion DPS running on G^u can halt and the solution returned is proven to be within the bound B. The reason is that DPS is a special case of Focal Search. By contrast DPSU running on G has no such guarantee to be within the bound B in terms of cost. This is because DPSU is not a Focal Search and may choose to expand nodes that are larger than $B \times f_{min}$. To remedy this, every time DPSU chooses a goal node for expansion, it performs a *goal cost test* to see whether $g(goal) \leq B \times f_{min}$. If $g(goal) > B \times f_{min}$ then DPSU continues to further expand nodes (according to $ud^u(\cdot)$) until a valid goal node is found. Thus, DPSU on G

[2] Sometimes $h^u(n)$ was called *distance-to-go* as opposed to the regular h-function which was called *cost-to-go* (Thayer and Ruml 2011). Due to ambiguity of these terms we chose not to use them.

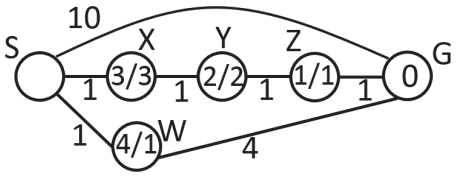

Figure 1: DPS vs. DPSU

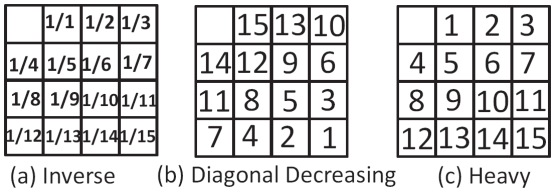

Figure 2: Versions of the weighted 15-puzzle

will expand at least as many nodes as DPS on G^u but it may possibly expand more nodes if the first goal reached is not under the bound with respect to the new real weighted cost.

We demonstrate the difference between DPS and DPSU through the example given in Figure 1 while assuming that $B = 2$. Heuristics inside nodes are in the format of h/h^u. After the start state is expanded, nodes G, X and W are generated. We note that at this point $f_{min} = f(X) = 4$ while $f^u_{min} = f^u(G) = 1$. First, let's consider DPS. $ud(G) = -\infty$, $ud(X) = \frac{B \times f_{min} - g(X)}{h(X)} = \frac{8-1}{3} = 2.333$ while $ud(W) = \frac{8-1}{4} = 1.75$. So, X is expanded and Y is generated with $ud(Y) = \frac{8-2}{2} = 3$. Next, Y is expanded and then Z is generated and expanded with $ud(Z) = \frac{8-3}{1} = 5$. Next G is generated, this time with $ud(G) = +\infty$. So, G is immediately expanded and the search halts. Nodes, S, X, Y, Z and G were expanded, a total of 5 nodes. Next, let's consider DPSU. Here, G is generated with $ud^u(G) = +\infty$. So, G is expanded and we have a solution which is within the bound in terms of number of edges. But, since $B \times f_{min} = 8$ and the real cost of the path is 10, the solution is not within the bound in terms of the real edge costs (this demonstrates that DPSU is not a Focal Search). Thus, DPSU continues its search. Now, $f^u_{min} = f^u(W) = 2$. Therefore, $ud^u(X) = \frac{B \times f^u_{min} - g^u(X)}{h^u(X)} = \frac{4-1}{3} = 1$ while $ud^u(W) = \frac{4-1}{1} = 3$. So, W is expanded and G is generated again with $ud^u(G) = +\infty$ but this time with a solution of cost 5. G is immediately expanded and the search halts because the solution is smaller than $B \times f_{min} = 8$. So, DPSU expanded 4 nodes, S, G, W and G and outperformed DPS.

4.1 Restricted DPSU

Restricted DPSU (RDPSU) is a version of DPSU that maintains FOCAL. RDPSU chooses the node with the best ud^u but only within FOCAL. Unlike DPSU, RDPSU is not allowed to expand nodes outside FOCAL. So, like any Focal Search, RDPSU can halt as soon as the first node is chosen for expansion. In the example above, G was first expanded with $ud^u = \infty$. So, DPSU chose to expand it only to realize that it is not within the bound. By contrast, RDPSU will not choose to expand G at this point as it is outside FOCAL and will save one node expansion. In total RDPSU expands only 3 nodes: S, W and then G after which it halts. The disadvantages of RDPSU is that it must maintain FOCAL in addition to OPEN. Similarly, RDPSU cannot choose to expand nodes outside FOCAL even if they may lead to a valid solution faster.

5 Experimental Results

The aim of this section is to study which algorithm should be used under what circumstances. We generated a number of weighted variants for known search benchmarks. We then experimented with DPSU, RDPSU and with state-of-the-art BSS algorithms DPS and *Explicit estimation search* (Thayer and Ruml 2011) (EES). EES is a BSS algorithm that uses both h and h^u (h^u was called d in the EES paper). In addition, EES also uses two other inadmissible estimates $\hat{h}$ and $\hat{d}$ which are supposed to be more accurate than h and h^u since they are not restricted to be admissible. EES is harder to implement than the DPS variants as it requires three OPEN lists and additional two inadmissable estimates. We did not compare with WA* (Pohl 1970) because it is usually weaker than EES and DPS (Gilon, Felner, and Stern 2016).

5.1 The 15-Puzzle

In the standard form of the 15-puzzle, each move costs 1. In the *heavy 15-puzzle* (Thayer and Ruml 2011) (labeled HEAVY) moving tile $\#X$ costs X. These weights are shown in Figure 2(c). By contrast, in the *inverse 15-puzzle* (labeled INVERSE) moving tile $\#X$ costs $\frac{1}{X}$ as shown in Figure 2(a). We can look at those versions as a powering factor of x (denoted α), which gives a flexible variant of weights from unit cost ($\alpha = 0$), HEAVY ($\alpha = 1$) or INVERSE ($\alpha = -1$). We also use this α parameter on the other domains seen below. Additionally we introduce a third version which we call the *heavy, diagonal-decreasing 15-puzzle* (HEAVY-DD). Similar to HEAVY, moving tile $\#X$ costs X. However, tiles in the goal state are distributed diagonally in such a way that large tiles are placed close to the top left corner and small tiles are placed far from it, as shown in Figure 2(b).

The Manhattan Distance heuristic (MD) for the 15-puzzle only counts moves of individual tiles but does not count internal conflicts between tiles. Naturally, these conflicts occur more for tiles with goal location that is close to the position of the blank in the goal state (top left position) as these tiles must perform the last moves in any solution. Weighted Manhattan Distance (WMD) generalizes MD by multiplying the MD of each tile by its weight. We note that for HEAVY, tiles that are close to the top left position have relatively low weights. Therefore the fact that MD misses many of their internal conflicts is not hurting WMD too much and WMD for HEAVY is relatively strong. By contrast, in INVERSE such tiles have relatively large weights. Therefore, missing their conflicts has a stronger effect on the accuracy of WMD and it is relatively weak for INVERSE. In HEAVY-DD tiles close

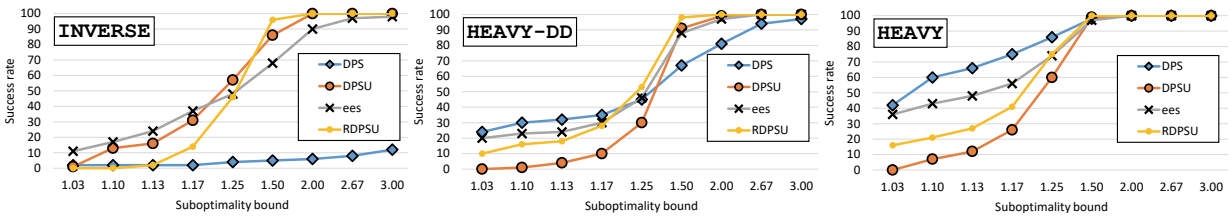

Figure 3: Success rate over the weighted 15 puzzle versions. Heavy (left), Heavy-DD (middle) and Inverse (right).

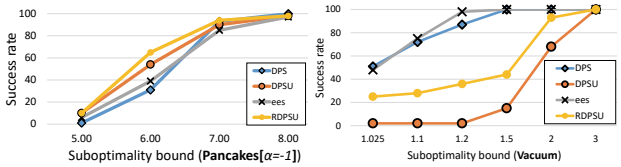

Figure 4: Success rate of the other domains.

to the top left position also have relatively large weights but the weights are decreasing linearly and not according to $1/x$. Thus, the effect on WMD is less severe than in INVERSE and WMD for HEAVY-DD is moderately inaccurate.[3]

As a result, DPS (and any other algorithm) that uses WMD will be relatively effective for HEAVY, less effective for HEAVY-DD and relatively weak for INVERSE. Nevertheless, as DPS degrades it will be more beneficial to use DPSU and RDPSU. In such cases, solving the problem on the abstract unit edge cost domain (thus using MD which is relatively more accurate than WMD) and migrating the solution back to the original problem is better than using the inaccurate heuristic of WMD.

We experimented on 100 random instances (Korf 1985) and varied the sub-optimality bound B. Each algorithm was halted when either a solution was found that is within the sub-optimality bound (success) or after 5,000,000 nodes were expanded without finding a solution under the bound (failure). The plots in Figure 3 show the *success rate*, i.e., the % of instances solved successfully for the three domains.

The results show different trends depending on the accuracy of the heuristic. In HEAVY, WMD is relatively accurate. DPS is therefore very effective and there is no need to exploit the abstract unit edge-cost solution nor its estimation h^u. Therefore, DPSU and RDPSU are weaker than DPS in HEAVY. In HEAVY-DD, WMD is moderately accurate and this weakens DPS. None of the algorithms excel in this domain as the heuristic is not strong enough to make DPS the winner (only h) and is not weak enough to make DPSU and RDPSU the winners (only h^u). Finally, in INVERSE WMD is weak. Therefore, DPS suffers greatly, and EES suffers but less because it also uses h^u. In INVERSE, it is beneficial to get the help of the abstract unit edge cost solution. Consequently, DPSU and RDPSU are the best algorithms here as they exclusively use h^u which is far more accurate (in the abstract domain) than WMD (in the original domain).

A clear trend in all three versions of the puzzle is that DPSU and RDPSU are very effective in high sub-optimality bounds – they are the best algorithm in all domains for $B \geq 1.5$ – and are relatively less strong for low values of B. This is reasonable, because finding valid solutions is more difficult as B decreases, and $ud^u(\cdot)$ provides no cost-aware guidance, as it completely ignores action cost (considering only h^u and g^u). However, as B grows, finding solutions under the bound is easier and thus ignoring the costs can focus the search towards finding solutions faster.

5.2 Other domains

Our next domain is the K-pancake puzzle. The task is to sort a vector of numbers $V[K]$ where there are $K-1$ operators, where operator i reverses a prefix of size $i+1$. As a heuristic, we used the GAP heuristic (Helmert 2010) which adds 1 for every two adjacent numbers that are not consecutive (hence a gap). We created a *heavy* variant of this puzzle where the cost of the operator that flips a prefix $(V[1] \dots V[i+1])$ is the maximum among the two elements on the extreme sides of the prefix, i.e., $\max(V[1], V[i+1])$. [4] Additionally we created a fine-tuned version with a parameter α, where we take $\max(V[1]^\alpha, V[i+1]^\alpha)$. To get the unit cost version one needs to set $\alpha = 0$. We used a version with $\alpha = -1$ on 101 pancakes. Results are provided in Figure 4 (left). The GAP heuristic is relatively weak for 101 pancakes. Therefore, DPS and EES are relatively weak and are outperformed by RDPSU and DPSU.

Finally we experimented on the *Vacuum Cleaner* introduced by Thayer and Ruml (2011) and is inspired by a state-space presented in Russell and Norvig's (1995) textbook. A vacuum cleaner is working in a grid (200×200) with obstacles (35% of the cells) and there are a number of dirt spots. The cleaner should find a tour that cleans all dirty spots. When carrying dirt, the cleaner becomes heavier and therefore every dirty spot that was cleaned adds 1 to the cost of moving the cleaner. The heuristic used is the *minimum spanning tree*. We also fine tuned this domain taking the cost to be $\#dirts^\alpha$ where α is a parameter. On this domain we used $\alpha = 0.5$. The results are shown in Figure 4 (right). Here the heuristic is relatively strong and this favors EES and DPS.

[3] We analyzed the accuracy of WMD in each of our 15-puzzle variants by measuring the heuristic error on a large sample of states. Our study confirms that WMD is relatively strong for HEAVY moderate for HEAVY-DD and relatively weak for INVERSE.

[4] The motivation is that the spatula should be big enough to hold the larger side of the flipped set of pancakes, or else it will topple.

6 Conclusions and Future Work

Our results show that no algorithm dominates in all circumstances. Relatively weak heuristics hurt DPS especially for large values of B. In such cases, RDPSU and DPSU are a better choice. But when the heuristic is strong there is no reason to move to the unit cost abstraction. Looking at all three domains we can clearly see that RDPSU tends to outperform DPSU especially for low values of B. In lower B values it is more important to be within FOCAL and RDPSU is restrictive in this sense.

In the future we intend to study whether this trend can be observed for other BSS algorithms (e.g., weighed A*) as well as for other settings (e.g. BCS) and for other domains.

7 Acknowledgments

This research was supported by Israel Science Foundation (ISF) grant #417/13 and by the Israel and Ministry of Science grant #8G15027.

References

Ebendt, R., and Drechsler, R. 2009. Weighted A* search - unifying view and application. *Artif. Intell.* 173(14):1310–1342.

Gilon, D.; Felner, A.; and Stern, R. 2016. Dynamic potential search - A new bounded suboptimal search. In *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, Tarrytown, NY, USA, July 6-8, 2016.*, 36–44.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *Third Annual Symposium on Combinatorial Search (SoCS)*.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.* 27(1):97–109.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Trans. Pattern Anal. Mach. Intell.* 4(4):392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI*, 12–17.

Russell, S., and Norvig, P. 1995. *Artificial intelligence: a modern approach*. Prentice Hall.

Stern, R.; Felner, A.; van den Berg, J.; Puzis, R.; Shah, R.; and Goldberg, K. 2014. Potential-based bounded-cost search and anytime non-parametric A*. *Artificial Intelligence* 214:1–25.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*.

Thayer, J. T.; Benton, J.; and Helmert, M. 2012. Better parameter-free anytime search by minimizing time between solutions. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*.

Valenzano, R. A.; Arfaee, S. J.; Thayer, J. T.; Stern, R.; and Sturtevant, N. R. 2013. Using alternative suboptimality bounds in heuristic search. In *ICAPS*.

Wilt, C. M., and Ruml, W. 2011. Cost-based heuristic search is sensitive to the ratio of operator costs. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011, Castell de Cardona, Barcelona, Spain, July 15.16, 2011*.

Wilt, C. M., and Ruml, W. 2014. Speedy versus greedy search. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014.*