Efficient Optimal Search under Expensive Edge Cost Computation

Masataro Asai *
Graduate School of Arts and Sciences
University of Tokyo
guicho2.71828@gmail.com

Akihiro Kishimoto, Adi Botea, Radu Marinescu,
Elizabeth M. Daly and **Spyros Kotoulas**
IBM Research Ireland
{akihirok, adibotea, radu.marinescu,
elizabeth.daly, spyros.kotoulas}@ie.ibm.com

Abstract

Optimal heuristic search has been successful in many domains, including journey planning, route planning and puzzle solving. Existing work typically assumes that the cost of each action can easily be obtained. However, in many problems, the exact edge cost is expensive to compute. Existing search algorithms face a significant performance bottleneck, due to an excessive overhead associated with dynamically calculating exact edge costs.

We present DEA*, an algorithm for problems with expensive edge cost computations. DEA* combines heuristic edge cost evaluations with delayed node expansions, reducing the number of exact edge computations. We formally prove that DEA* is optimal and it is efficient with respect to the number of exact edge cost computations.

We empirically evaluate DEA* on multiple-worker routing problems where the exact edge cost is calculated by invoking an external multi-modal journey planning engine. The results demonstrate the effectiveness of our ideas in reducing the computational time and improving the solving ability. In addition, we show the advantages of DEA* in domain-independent planning, where we simulate that accurate edge costs are expensive to compute.

1 Introduction

Heuristic search can tackle many difficult problems, including puzzles, journey planning and route planning e.g., (Korf 1985; Botea, Nikolova, and Berlingerio 2013; Sturtevant et al. 2015). Algorithms such as A* (Hart, Nilsson, and Raphael 1968) can be used to find optimal solutions. Existing optimal heuristic search algorithms typically assume that edge costs are known *a priori*. For example, the unit edge cost of 1 is used to find the smallest number of moves to solve a sliding puzzle instance. In the Traveling Salesperson Problem (TSP), the cost of an edge typically is fixed, and edge costs are obtained with a negligible computational overhead (e.g., using table lookups).

However, in some domains, it is difficult to pre-compute the exact edge costs. They must be computed dynamically, during search. When dynamic edge cost computations are expensive, optimal heuristic search faces a major challenge stemming from an excessive overhead caused by edge cost computations.

Consider the TSP in a setting where a city's multi-modal transport network is used for travel. The travel time between two locations can depend on factors such as the departure time and uncertain events such as traffic jams. Multi-modal journey planning can provide travel times between two locations, with a given starting time, taking into account uncertainty, such as potential missed connections due to variations in the arrival and departure bus times (Botea, Nikolova, and Berlingerio 2013).

Despite recent advances in optimal multi-modal journey planning under uncertainty (Botea and Braghin 2015; Kishimoto, Botea, and Daly 2016), computing the optimal travel time can be expensive. For instance, Kishimoto, Botea, and Daly (2016) showed cases where their journey planner needs 100 seconds and visits 1 million nodes to compute one journey plan between two locations for a given departure time. In addition, the optimal travel time between two locations cannot be precomputed and stored in a table due to prohibitively many combinations of an origin, a destination, and a departure time.

We present Delayed Expansion A* (DEA*), an A*-based algorithm that efficiently performs dynamic edge cost computations, during heuristic search. Using admissible estimates of exact edge costs, DEA* can delay the computation of exact edge costs. This way, DEA* reduces the number of exact edge computations while always producing optimal solutions. Besides the main DEA* algorithm, we describe several search enhancements including dominance-based pruning, and caching.

Part of the evaluation focuses on a problem called Multiple-Worker Routing Problem (MWRP). We introduce MWRP and prove that is is an NP-hard problem. Besides MWRP, we demonstrate our ideas in domain-independent planning problems, where we simulate that exact edge costs might be expensive to compute. Our results clearly show that our approach significantly reduces the computational time and improves the solving ability.

2 Delayed Expansion A*

In this section, we present our algorithm DEA*. Consider a problem where edge costs are expensive to compute. As said earlier, examples could include edges representing a travel

*This work is done in IBM Research Ireland. He is also supported by a JSPS Grant-in-Aid.

Algorithm 1 DEA*

Input: n_0

Initialize OPEN $= \emptyset$, $g(n_0) = 0$, $(\forall n \neq n_0;\ g(n) = \infty)$
push(n_0, OPEN), Mark s_0 as standard (i.e., not temporary)
while OPEN $\neq \emptyset$ **do**
  $n = \text{pop(OPEN)}$
  if n has a duplicate $n' \in$ CLOSED and $g(n') \leq g(n)$ **then**
    Continue
  else if n is temporary **then**
    $g(n) \leftarrow g(\text{parent}(n)) + c_a(\text{parent}(n), n)$
    Adjust $s(n), h(n), f(n)$ based on c_a
    push(n, OPEN), Mark n as standard (i.e., not temporary)
    Continue
  else
    Add n to CLOSED, Update $g(n)$ in CLOSED if necessary
  if n is a goal **then**
    Extract and return solution
  Generate successors(n) based on c_h
  for each $m \in$ successors(n) **do**
    $g_h = g(n) + c_h(n, m)$
    $g(m) \leftarrow g_h$, parent$(m) \leftarrow n$
    push(m, OPEN), Mark m as temporary

leg, from an origin to a destination, in a multi-modal travel network characterized by uncertainty. If A* is used to solve such a problem, every time when A* traverses[1] an edge in the search graph, the exact cost of that edge has to be available. A* could compute these costs on demand, and cache the results for a future reuse.

However, caching and reusing actual costs has limited usefulness. At the same time, heuristic admissible estimations for travel times can often be provided much more quickly. DEA* takes advantage of this, using heuristic estimates of edge costs to delay or even avoid entirely the computation of an accurate cost. Often, an optimal solution is found with no need to compute accurate costs for all generated nodes. This is where the advantage of DEA* stems from.

In the rest of this section we describe DEA*, provide an example and perform a theoretical analysis of DEA*.

2.1 The DEA* Algorithm

Given a search node n, assume that n contains state information $s(n)$, a pointer to the parent node, a g-value $g(n)$, a h-value $h(n)$ and an $f = g + h$ value. In DEA*, we distinguish between two types of edge costs, namely an admissible heuristic estimation c_h and an actual cost c_a.

Consider an edge (p, n) from a parent node to a successor node. Using a heuristic cost for this edge could impact $g(n)$, $s(n)$, $h(n)$ and $f(n)$, as follows. By definition, g is impacted, as g is the sum of all edge costs of the path available from the root to n. To show the potential impact on $s(n)$, consider a problem where some edges encode traveling legs from one location to another. Assume further that the arrival time at a current location is part of the state definition. Clearly, the arrival time can depend on the cost of the incoming edge, i.e., on the travel time from a previous location to the current location. The heuristic $h(n)$ could be impacted because it typically is a function of the state information $s(n)$.

[1]We say that A* traverses an edge if that edge is the transition from a node currently being expanded to a successor.

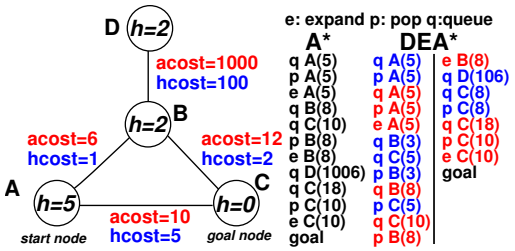

Figure 1: Example illustrating that standard A* computes an unnecessary actual edge cost of BD ($c_a = 1000$). In the figure, "acost" stands for c_a and "hcost" stands for c_h.

Algorithm 1 shows DEA* in pseudocode. DEA* is built on top of A*. For clarity, the pseudocode is based on a simple variant of A*. A newly generated node n could be discarded straight away if a duplicate n' exists in CLOSED with $f(n') < f(n)$. The pseudocode does not show this, for simplicity.

In DEA*, when a node p is expanded (and thus a successor n is generated), only the heuristic cost c_h of the edge (p, n) is computed. Node data such as $g(n), s(n), h(n), f(n)$ are computed with the heuristic cost c_h in use for the edge (p, n).

After being generated, n is enqueued into the OPEN list based on the heuristic cost c_h. A node enqueued into the OPEN in this fashion is called a *temporary node*. When a temporary node n is popped from OPEN, the actual cost c_a of the edge from its parent is computed, and $g(n), s(n), h(n)$ and $f(n)$ are updated accordingly. The node changes its status from a temporary node into a standard node. The node is re-inserted into the OPEN list with the new f value, and the main cycle of DEA* continues.

2.2 Example

Fig. 1 illustrates differences between DEA* and A* on a toy, 4-node search graph. A* expands the nodes A, B and C in this order, during which it evaluates the actual cost of edges AB, AC, BD and BC in this order. The computation of BD, which yields a value of 1000, is unnecessary, since all other edge costs are significantly smaller.

DEA* does not compute the actual cost c_a of edge BD. After the root node A is expanded, B and C are enqueued with f values based on c_h estimations of the edges from their parent. Then B, whose f value is 3, is popped from the OPEN list. Now the actual cost c_a of the edge AB is computed, and $f(B)$ becomes $6+2 = 8$. In this simple example, we assume that the h value of a node stays constant (i.e, it does not change when a node changes its status from temporary to permanent). Node B is re-inserted into the OPEN, and node C is popped out. Its f value increases from 5 to 10, which triggers its re-insertion into the OPEN, with the new f value. The process continues as shown in Fig. 1, with node D never being popped from the OPEN.

As illustrated in Fig. 1, DEA* can perform more OPEN list operations than standard A*. However, in scenarios

where edge cost computations, or even state heuristic computations are expensive, OPEN list operations have a much smaller impact on the overall running time.

2.3 Theoretical Analysis

We prove two theoretical properties of DEA*.

Theorem 1. *DEA* outputs optimal solutions.*

Proof. Sketch: DEA* expands only regular nodes, i.e., nodes computed with c_a in use.

Given DEA*, consider an underlying A* that breaks ties among nodes in the OPEN in the same way as DEA*. DEA* and the A* under consideration expand the same set of nodes, in the same order.

Based on the two facts above, it follows that DEA* and the corresponding A* compute identical solutions. We omit a more detailed proof due to space limitations. □

Given a node n, let n_h be its temporary version, and n_a its standard version. Let p be the parent node. Note that p is already a standard node at the time when n is generated, since node are expanded only after they become standard nodes. Define $g(n_a) = g(p) + c_a(p, n_a)$, and $g(n_h) = g(p) + c_h(p, n_h)$. We say that the heuristic h is consistent with respect to c_h if, for every parent-child pair (p, n), $h(p) \leq c_h(p, n_h) + h(n_h)$ and $h(G) = 0$ for every goal node G. Assuming further that $h(n_h) \leq h(n_a), \forall n$, if h is consistent with respect to c_h, it is also consistent with respect to c_a (i.e., standard consistency).

Theorem 2. *Assume that A* and DEA* use a similar tie breaking scheme, and that they use a heuristic h that is consistent with respect to c_h. DEA* cannot generate more nodes with actual edge costs than A*.*

Proof. Let c^* be the optimal cost of a solution. The set of nodes that DEA* generates using c_a includes the set S_1 of all nodes n with $f(n_h) < c^*$, plus S_2, a subset of the nodes n with $f(n_h) = c^*$, depending on tie breaking. The set of nodes generated by A* includes the set S_3 of all nodes n with $f(\text{parent}(n)) < c^*$, plus S_4, a subset of all nodes n with $f(\text{parent}(n)) = c^*$, depending on tie breaking.

Let n be a node and let p be its parent. Then $f(p) = g(p) + h(p) \leq g(p) + c_h(p, n_h) + h(n_h) = f(n_h)$.

It follows that $S_1 \subseteq S_3$ and $S_2 \subseteq S_4$, assuming a similar tie-breaking scheme. □

3 The Multiple-Worker Routing Problem

MWRP is a realistic scheduling problem that requires finding an optimal schedule for a set of workers to deliver services to a set of customers residing at different fixed locations. For simplicity and without loss of generality, we consider a simple healthcare scenario where the customers are *patients* and the workers deliver at-home care services such as nursing care, medical care, and physical therapy. Each patient has an appointment time when a worker should arrive and perform a healthcare task. To visit and treat patients, a worker uses the multi-modal public transport available in a city. Emergency situations, where using an ambulance is more appropriate, are beyond our focus. So is the rescheduling and the dynamic assignment or reassignment of tasks.

In a valid solution, all patients have to be treated. Tasks can be performed later than their appointment time, but each delay in performing a task degrades the quality of a solution. The cost of a solution is the sum of all delays of all tasks. A task starting at its planned appointment time has no delay (i.e., the delay is 0). Tasks starting later have a positive delay. A task cannot start earlier than their planned appointment time (i.e., even if the worker arrives early, they will have to wait until the task is supposed to start).

More formally, a four-tuple $\langle P, A, W, \tau(x, y, t)\rangle$ defines an MWRP, where $P = \{p_i | 1 \leq i \leq n\}$ is a list of locations of the patients, $A = \{a_i | 1 \leq i \leq n\}$ is a list of appointment times, $W = \{w_j | 1 \leq i \leq m\}$ is a list of initial locations of workers and $\tau(x, y, t)$ is a travel time function between two locations $x, y \in P \cap W$ with the start time $t \in [0, 86400]$.[2]

It is important to note that in practice the travel time depends on the start time, the schedules of the public transport vehicles, and the other factors such as uncertainty about the actual arrival and departure times of buses.

We show that solving MWRP optimally is NP-hard. We define the MWRP-OPT problem as follows. The input is an MWRP instance and a number $k \geq 0$. The question is whether the instance has a solution whose cost (delay) D satisfies $D \leq k$.

Theorem 3. *MWRP-OPT is NP-hard.*

Proof. We show this with a reduction from the single machine total tardiness problem (SMTTP), an NP-hard problem (Du and Leung 1990). Given a set of jobs $\{1, 2, \cdots, n\}$ to be processed on a single machine and the processing time p_i and due date d_i of each job i, the objective is to find a schedule for the jobs that minimizes the tardiness $T = \sum_{i=1}^{n} T_i$, where $T_i = \max(0, C_i - d_i)$ and C_i is the completion time of job i according to the schedule.

Consider an arbitrary SMTTP instance defined as above. We construct an MWRP instance in polynomial time as follows. There is one worker and n patients. The appointment time of patient i is d_i, and the duration of the appointment (treatment) is 0. The worker walks between locations, and the travel time from location j ($j \neq i$) to location i is p_i.

It is easy to see that the tardiness T in the SMTTP instance is equal to the total delay (i.e., the cost) D of the MWRP instance. Thus, the SMTTP instance has a solution with $T \leq k$ iff the MWRP instance has a solution with $D \leq k$. □

State Space Representation

We next define the state space used for solving MWRP. A search state is represented by 3-tuple $\langle B, T, L\rangle$ where $B = \{b_i | 1 \leq i \leq n\}$ is a bit vector representing whether a patient i has already been treated, $T = \{t_j | 1 \leq i \leq m\}$ is a list of current times of the workers and $L = \{l_j | 1 \leq i \leq m\}$ is a list of locations of workers at time t_j.

We define a single action schema, denoted by *treat*(i, j), where i is a patient and j is a worker. A *treat* action has the following effects. The bit b_i corresponding to patient

[2] $86400 = 24 \times 60 \times 60$ is the number of seconds in a day.

i is set to true. The current time t_j of worker j is updated as follows. As tasks do not start before the appointment time a_i, the worker's time is set to either the arrival time, or the appointment time, whichever comes later: $t_j = \max(t_j + \tau(l_j, p_i, t_j), a_i)$. Finally, the location l_j of worker j is updated to the location p_i of the patient i.

The cost of an action is defined by the *delay*, namely $\max(t_j + \tau(l_j, p_i, t_j) - a_i, 0)$. This means that the delay is 0 when the worker arrives in time or earlier than the appointment time. Otherwise, the delay is positive.

A solution to an MWRP instance $\mathcal{M}$ is a sequence of actions *treat*(i, j) where every appointment is addressed with a *treat* action. The solution cost is the sum of the action delays. An optimal solution minimizes the total delay.

In this MWRP setting, the travel time between two location (i.e., $\tau(\cdot)$) is *not* given *a priori*. In practice, the actual travel time depends on multiple factors such as the origin, the destination, the time of travel, and the uncertainty about the multi-modal transportation network (e.g., exact arrival and departure times for buses). Pre-computing accurate values for all travel times often is impractical, due to many combinations of origins, destinations, and departure times.

4 Search Enhancements in MWRP

We present a pruning strategy based on state dominance, an admissible heuristic h, and other implementation details.

State dominance We assume the following monotonicity over the arrival time of a trip:

Assumption 1. *For any start time $t_1 \leq t_2$, it holds that $t_1 + \tau(x, y, t_1) \leq t_2 + \tau(x, y, t_2)$.*

This implies the following property:

Proposition 1. *The delay $d(i, j) = \max(t_j + \tau(l_j, p_i, t_j) - a_i, 0)$ is monotonically increasing over t_j.*

Given that the cost of a path is a sum of delays, we get the following dominance criterion:

Definition 1. *For two states s_1 and s_2, we say that s_1 dominates s_2 (denoted as $s_1 \leq s_2$) when $\forall j, t_j(s_1) \leq t_j(s_2)$ and $\forall j, l_j(s_1) = l_j(s_2)$ and $\forall i, b_i(s_1) = b_i(s_2)$.*

For such states, the cost of an optimal path containing s_1 is no greater than the cost of an optimal path containing s_2. Thus it is safe to prune dominated states without losing the optimality of the solution computed.

Note that DEA* does not apply the dominance relationship when the dominating node is a temporary node. This is because the current time of the temporary node could be updated to a larger value when the node is reevaluated (i.e., when the node is converted into a standard node).

Admissible heuristic We have implemented a variant of the h^2 heuristic (Haslum and Geffner 2000), adapted to MWRP. Since the computation of h^2 requires the edge costs of all applicable actions available at the current state, a straightforward implementation results in extensive computations of accurate travel costs. We address this bottleneck by using heuristics costs c_h. That is, the computation of the h^2 heuristic uses heuristic estimations of travel times, instead of accurate travel times.

In our implementation, both accurate travel times and heuristic travel times are obtained with the DIJA journey planning system (Botea, Nikolova, and Berlingerio 2013). The heuristic estimations are computationally cheaper than the actual travel times. The results of invoking DIJA, both for heuristic and actual times, are cached for a future reuse.

DIJA pre-computes a lookup table of admissible heuristic estimates of travel times (Botea, Nikolova, and Berlingerio 2013). The system can combine deterministic and non-deterministic search (Kishimoto, Botea, and Daly 2016). Specifically, it can run a deterministic A* that computes the travel time for an optimistic, *best-case* scenario (e.g. buses arrive in time). Deterministically computed travel times can be used as an admissible heuristic for an AO* search that computes an uncertainty-aware, contingent plan. We use the search result of DIJA's A* to compute heuristic travel times (c_h). We denote this heuristic travel time by $\tau'(x, y, t)$. The function τ' also satisfies the admissibility and monotonicity criteria that hold for τ in Proposition 1.

The h^2 heuristic is computed as follows: For each of every combinations of two subgoals (treating the corresponding patients), it computes the minimum cost sum (the sum of delays) for achieving them, and returns their maximum. The cost is calculated based on the *hcost* c_h. Since $c_h \leq c_a$, h^2 heuristics calculated with c_h is a lower bound of h^2 heuristics calculated with c_a, which is in turn a lower bound of the total cost from the current state to a goal state. Thus, the h^2 heuristic computed with c_h is admissible.

As mentioned in Sec. 2, amending the cost of the edge from the parent can impact the state, which contains the current time of each worker. This change in the state can also trigger a change in the heuristic value of that node. Thus, heuristic functions might have to be computed twice for a node: once when the node is temporary, and once after the node becomes a standard node.

We can further enhance DEA* with a combination of the $h^{\max}$ (aka the h^1 heuristic) and h^2 heuristics. Temporary nodes are evaluated with h^1, and standard nodes by h^2. We chose this enhancement because it can be trivially implemented in DEA*. In contrast, A* needs a significant modification to the algorithm in order to adopt this approach.

Finally, A*'s performance is often affected by the tie-breaking strategy (Asai and Fukunaga 2016). Ties on the f value are broken by preferring states with fewer untreated patients, and further ties are broken in the first-in-last-out order.

Other implementation details DIJA's AO* search produces an optimal contingent plan. In a DIJA optimal contingent plan, there is one *safe* pathway, and zero or more *opportunistic* pathways. All actions along the safe pathway can be executed with a probability of 1, if the traveler decides to follow that pathway. The safe pathway is the slowest (i.e., has the largest travel time) among all pathways in the optimal contingent plan. Opportunistic pathways are faster, but there is no guarantee that the traveler will be able to follow them at the plan execution time.

For the actual edge cost function c_a we use the travel time of the safe pathway. Other options for implementing c_a, such

city	nodes	segments	stops	routes	trips/day
Dublin	301,638	319,846	4,739	120	7,308
Montpellier	152,949	161,768	1,297	36	3,988
Rome	522,529	566,400	8,896	391	39,422

Table 1: Statistics of transport data used in experiments.

City	$\|W\|$	Blind		$h^{\max}$		h^2		h^1h^2
		A*	DEA*	A*	DEA*	A*	DEA*	DEA*
Total		50	**59**	68	**77**	81	**83**	**85**
Dublin	12	14	**15**	15	**21**	19	**22**	**22**
	6	2	**4**	8	8	12	12	11
Montpellier	12	16	15	17	17	17	15	**18**
	6	1	**4**	3	**5**	3	**4**	**5**
Rome	12	15	**18**	18	**19**	19	19	**20**
	6	2	**3**	7	7	11	11	9

Table 2: Number of solved instances in MWRP. DEA* scores are highlighted when they improve upon A*.

as using the expected arrival time, are left as future work.

5 Experimental Results

We evaluate DEA*'s performance in the MWRP and in domain-independent planning. Experiments are conducted on an Intel Xeon CPU cluster, with a time and memory limit of one hour and 1.5 million in-memory states per instance.

Evaluation in the MWRP We generated a total of 180 instances with the real road-map and transportation data from three European cities (Table 1). In each instance, there are either 6 or 12 workers, while the number of patients was varied between 8–24. They do not contain unrealistic configurations (e.g. 6 workers & 24 patients means that each worker attends 4 patients per hour on average). The locations of workers W and the locations of patients P are randomly selected within a 2km radius circle in the city center. Appointment times are randomly set between 10–11AM. Workers' start times are fixed at 10AM.

Our evaluation includes three heuristics used in DEA*: blind, $h^{\max}$ (aka h^1), and h^2. Compared to h^2, $h^{\max}$ is faster but less informative, since it returns only the maximum of the minimum possible delay for each untreated patient. Table 2 shows the number of instances solved by A* and DEA*. In terms of total coverage, DEA* always solves more instances than A*, when they both use the same heuristic. That is, a blind DEA* solves 9 more instances than a blind A*, DEA* with h^1 solves 9 more instances than A* with h^1, and DEA* with h^2 solves 2 extra instances as compared to A* with h^2.

Moreover, the combined use of h^1 and h^2 enables DEA* to solve two additional instances compared to DEA* (h^2), achieving the best solving ability (see h^1h^2 in Table 2). The results clearly show the importance of our delayed node evaluation in the presence of expensive edge cost computations.

The histogram shown in Fig. 2 indicates the ratio of the runtime spent by the external DIJA engine to the total time of A* + DIJA. Despite integrating caching, the edge cost computations are the main performance bottleneck in A*. In most cases, irrespective of the heuristics used, calls to DIJA cover more than 90% of the runtime of A* + DIJA.

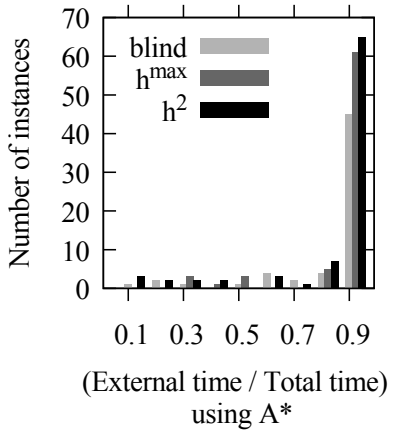

Figure 2: Histogram of the ratio of the external DIJA runtime to the total runtime in A*.

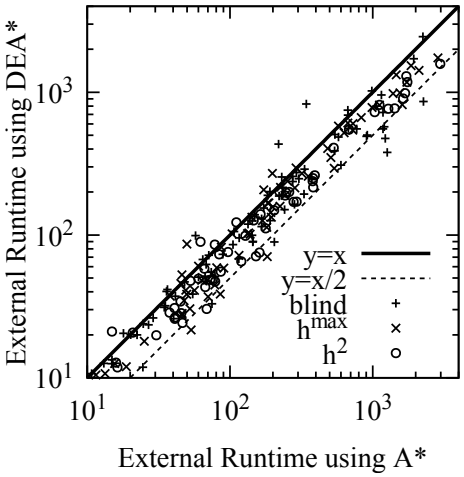

Figure 3: Comparison of the runtime spent for the external calls DIJA, between A* and DEA*, on instances solved by both.

Fig. 3 plots the runtime spent on the external calls to DIJA, for the instances solved by both DEA* and A*. The time spent by A* is plotted on the vertical axis against DEA* on the horizontal axis on logarithmic scales. Points below the $y = x$ line indicate that DEA* outperforms A*. These figures clearly show that DEA* successfully reduces the number of expensive calls to DIJA. The average improvement to the total runtime was 54%(blind), 123%($h^{\max}$), 149%(h^2).

Note that these two charts do not show the best performing version of DEA*, namely DEA* with h^1h^2. The reason is that in these charts we wanted to compare DEA* vs A* when both algorithms use the same heuristic.

Evaluation in Domain Independent Planning We consider the case where domain independent classical planning needs to account for the expensive edge cost evaluation. Current domain independent planners support the cases where the exact edge costs are obtained easily. For this reason, we simulate that accurate edge costs might be computationally expensive to obtain, and that a cheaper heuristic estimation

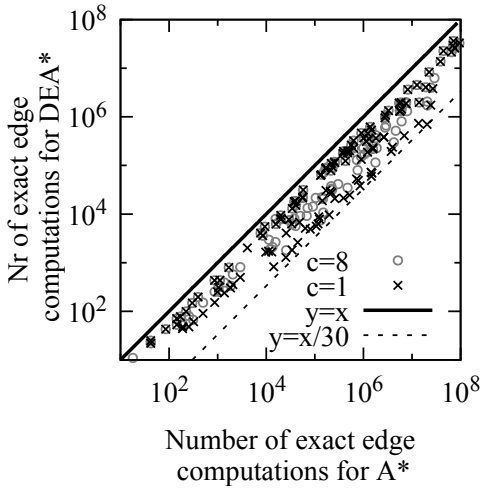

Figure 4: DEA* vs A* in domain independent planning.

can be computed.

We implemented DEA* on top of the cost-optimal Fast Downward planner based on the landmark-cut heuristic (Helmert and Domshlak 2009). We selected domains with non-unit action costs from the optimal tracks in past International Planning Competitions. We then assumed that Fast Downward can obtain the heuristic edge cost c_h by subtracting a constant c from the actual edge cost c_a, i.e. $c_h = \max(0, c_a - c)$, as inspired from (Helmert and Röger 2008).

We selected the following non-unit action cost domains: barman, cybersec, elevators, floortile, openstacks, parc-printer, pegsol, scanalyzer, sokoban, transport and woodworking, and considered 20 instances in each domain. We ran the solver with a time and memory limit of 30 minutes and 4GB per instance. h^{LMcut} is computed based on c_h, similarly to the heuristics h^1 and h^2 implemented in MWRP.

In Fig. 4, the number of c_a calculation required by A* for each instance is plotted on y-axis, against DEA* on x-axis, on logarithmic scales. We show results obtained with $c = 8$ and $c = 1$, and DEA* significantly reduces the number of exact edge cost computations. In terms of improvement factor (i.e. x/y), when $c = 8$, the minimum factor is 1.6, the mean 4.5 and the maximum 18. For $c = 1$, the improvements are even greater. The minimum improvement factor is 1.7, the mean factor is 6.5 and the maximum one is 29.

6 Related Work

DEA* is related to Partial Expansion A* (PEA*) (Yoshizumi, Miura, and Ishida 2000) and Enhanced PEA* (EPEA*) (Goldenberg et al. 2014). However, these aim at different types of improvements. PEA* addresses high memory requirements caused by a large branching factor. For example, in the Multiple Sequence Alignment (MSA), the flagship application domain of PEA*, the number of successors per state is $O(2^d)$ where d is the number of sequences to be aligned. PEA* *generates and evaluates all successors* of a search state but keeps in memory only *a partial subset of successors* that look promising. As PEA* evaluates every successor, it requires that exact edge costs are immediately available and thus node evaluations are relatively cheap.

	Edge cost / g-value evaluation	Successors added into OPEN
A*	All successors, exact	All successors
PEA*	All successors, exact	**Partial subset**
DEA*	All successors, **heuristic**	All successors

On the other hand, DEA* addresses the issue of expensive exact edge cost computations required for obtaining the g-value. When a node n is generated, DEA* uses a heuristic estimation of the cost of the edge from its parent. The exact cost computation of that edge is delayed until n is selected for expansion.

Phillips, Likhachev, and Koenig (2014) present PA*SE, a parallel A* for robot motion planning domains. PA*SE is designed for domains where generating successor states can be expensive. E.g., in motion planning problems successor generation may involve an expensive and precise collision checking procedure. In contrast, DEA* is designed for domains where generating successors is not expensive, but computing the exact cost of a parent–successor edge could be.

Deferred evaluation (DE) has been studied in satisficing domain independent planning (Helmert 2006; Richter and Helmert 2009). When generating a successor of a node, DE sets the heuristic value of the successor to that of the parent, thus deferring the successor evaluation. As opposed to DEA*, this related work assumes that edge costs are readily available, and suboptimal solutions are allowed.

Decentralized, auction-based multi-agent coordination has been studied in multi-robot routing problems (Kishimoto and Sturtevant: 2008; Kishimoto and Nagano 2016). Edge costs are dynamically computed via path-finding on road or game maps. The similarity to DEA* is that edge costs can be expensive to compute. A key difference is that auction-based techniques are suboptimal.

7 Conclusions

Heuristic search algorithms typically assume that the cost of edges in the search graph are readily available. However, in real-life domains, such as problems that involve traveling between various locations on a map, such a simplifying assumption does not always hold.

We have introduced DEA*, a search algorithm based on A*, capable to reduce the number of exact edge cost computations in a search. We formally introduced the Multiple Worker Routing Problem, a practical application domain inspired from the healthcare industry. We provided an NP-hardness result for MWRP. We evaluated the performance of DEA* in MWRP and in domain-independent planning. The results demonstrate a significant performance improvement of DEA* as compared to A*.

As an interesting avenue of future work, Factored Planning (Amir and Engelhardt 2003; Brafman and Domshlak 2006; Asai and Fukunaga 2015) could incorporate ideas from DEA*. This framework automatically decomposes the

input problem and solves each subproblem in order to convert the subplans into macro actions. DEA* could improve this potential bottleneck by replacing the subproblem solving with a partial computation of the plan.

References

Amir, E., and Engelhardt, B. 2003. Factored Planning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Asai, M., and Fukunaga, A. 2015. Solving Large-Scale Planning Problems by Decomposition and Macro Generation. In *Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Asai, M., and Fukunaga, A. 2016. Tiebreaking Strategies for Classical Planning Using A^* Search. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Botea, A., and Braghin, S. 2015. Contingent versus Deterministic Plans in Multi-Modal Journey Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS)*, 268–272.

Botea, A.; Nikolova, E.; and Berlingerio, M. 2013. Multi-Modal Journey Planning in the Presence of Uncertainty. In *Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Brafman, R. I., and Domshlak, C. 2006. Factored Planning: How, When, and When Not. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Du, J., and Leung, J. 1990. Minimizing Total Tardiness on One Machine is NP-Hard. *Mathematics of Operations Research* 15:483–495.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N. R.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced Partial Expansion A*. *J. Artif. Intell. Res.(JAIR)* 50:141–187.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Haslum, P., and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proceedings of International Conference of AI Planning Systems*.

Helmert, M., and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Helmert, M., and Röger, G. 2008. How Good is Almost Perfect? In *Proceedings of AAAI Conference on Artificial Intelligence*.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)* 26:191–246.

Kishimoto, A., and Nagano, K. 2016. Evaluation of Auction-Based Multi-Robot Routing by Parallel Simulation. In *Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS)*, 495–503.

Kishimoto, A., and Sturtevant:, N. 2008. Optimized Algorithms for Multi-Agent Routing. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1585–1588.

Kishimoto, A.; Botea, A.; and Daly, E. 2016. Combining Deterministic and Nondeterministic Search for Optimal Journey Planning Under Uncertainty. In *ECAI 2016: 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands-Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285, 295. IOS Press.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 27(1):97–109.

Phillips, M.; Likhachev, M.; and Koenig, S. 2014. PA* SE: Parallel A* for Slow Expansions. In *Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Richter, S., and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS)*, 273–280.

Sturtevant, N. R.; Traish, J. M.; Tulip, J. R.; Uras, T.; Koenig, S.; Strasser, B.; Botea, A.; Harabor, D.; and Rabin, S. 2015. The Grid-Based Path Planning Competition: 2014 Entries and Results. In *Proceedings of the 8th Annual Symposium on Combinatorial Search*, 241–251.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with Partial Expansion for Large Branching Factor Problems. In *AAAI/IAAI*, 923–929.