A Framework for Quasi-Exact Optimization using Relaxed Best-First Search

Rüdiger Ebendt and Rolf Drechsler

Institute of Computer Science, University of Bremen
28359 Bremen, Germany
{ebendt,drechsle}@informatik.uni-bremen.de

Abstract. In this paper, a framework for previous and new quasi-exact extensions of the A^*-algorithm is presented. In contrast to previous approaches, the new methods guarantee to expand every state at most once if guided by a so-called monotone heuristic. By that, they account more effectively for aspects of run time while still guaranteeing that the cost of the solution will not exceed the optimal cost by a certain factor. First a general upper bound for this factor is derived. This bound is $(1+\epsilon)^{\lfloor \frac{N}{2} \rfloor}$ where N is (an upper bound on) the maximum depth of the search. Next, we look at specific instances of the algorithm class described by our framework. For one of the new methods a linear, i.e. much tighter upper bound is obtained: the cost of the solution will not exceed the optimal cost by a factor greater than $1+\epsilon$. The parameter $\epsilon \geq 0$ can be chosen by the user. Within a range of reasonable choices for ϵ, all new methods allow the user to trade off run time for solution quality. Besides that, the formal framework also serves for a comparison in terms of other algorithmic properties of interest, e.g. in terms of a necessary condition for state expansion.
The results of experiments targeting the minimization of *Binary Decision Diagrams* (BDDs) demonstrate large reductions in run time when compared to the best known exact approach for BDD minimization and to previous relaxation methods. Moreover, the quality of the obtained solutions is often much better than the quality guaranteed by the theory.

1 Introduction

In many real-world problems, dominating effort is spent on search, often involving huge state spaces. Therefore in the past many researchers have proposed heuristic and exact search algorithms. The drawbacks of blind methods are overcome by *heuristic search* methods to guide the search on a state space: with every state q a quantity $h(q)$ is associated which allows to search in the direction of the goal states. A prominent guided search algorithm is the well-known A^**-algorithm* [6]. A^* can be devised to find the minimum cost path in a graph describing the possible transitions from one state to another, i.e. in a state space graph. In its original form, A^* guarantees to find an optimal solution and it is used in many fields of application, including diverse areas such as robotics [7] and logic synthesis [5]. Hereby, two components of information are used with every state q: one is $g(q)$, which is the information about the cost of the path already covered. The other is the heuristic function $h(q)$, an estimate of the least cost of the remaining part. The first information, $g(q)$, adds a breadth-first component to the search while the second, $h(q)$, can devise the search to delve deeper into certain paths when they seem promising, i.e. it adds a depth-first component to the search. A^* searches the state space by systematically expanding the most promising state (i.e., a best-first search is performed) and generating states until a match to a goal condition is found. For this purpose a prioritized list OPEN orders the search within the states that are eligible for expansion, and closed states are maintained on a list CLOSED. If certain requirements to the heuristic function guiding the search are met, A^* will find a minimum cost path to a goal state [6].

A serious drawback of A^* is that, in the worst case, the run time as well as the amount of memory required to store OPEN and CLOSED is exponential in the depth of the search. This has led to several extensions of A^*, some of which are memory bounded, e.g. [15], while others mainly target to reduce the run time by allowing for bounded sub-optimality, i.e. they provide A^*-based quasi-exact approximation methods.

E.g., the idea of *Dynamic Weighting* (A^*_{DW}) [10] is to start with a high weighting of the depth first component at the beginning of the search (as this may help to find a promising direction more quickly) and then dynamically weigh the depth-first component less heavily as the search goes deeper (as this may help to prevent too early, i.e. premature termination). In contrast to that, in [9], the *Traveling Salesman Problem* (TSP) has been tackled by an extension of A^* called A^*_ϵ which relaxes the *selection condition* of A^*. This condition triggers the choice of the next state for expansion (i.e. for generating all its successors). More recently, a conceptually much simpler idea has been used in [16]: here, the depth-first component is constantly inflated by a certain factor $1+\epsilon$, $\epsilon > 0$. It has also been embedded in a so-called *Anytime Repairing* variant of A^* (ARA^*) in [7]. This idea is referred to as $A^*_\uparrow$ throughout the paper.

The contributions of the paper are twofold: On the one hand, a formal analysis aims at deeper theoretical insight and, based on the formal results, new improved methods are devised. On the other, the new methods are compared with each other and with previous methods during an experimental evaluation. First, a formal framework provides a unifying view that describes all of the above mentioned approaches. With the help of this framework, all methods can be identified as special instances of one generic relaxation algorithm. This is of particular interest since the ideas of the methods may seem very different at first glance. As a next step, several interesting properties that are shared among all considered methods can be directly deduced. These are the so-called ϵ-admissibility and a necessary condition for state expansion: the first guarantees that the deviation of the solution must be bounded by $1+\epsilon$, the second one is important for efficiency considerations.

When searching in large state spaces, a potential source of performance loss is the repeated consideration of the same states, i.e. so-called reopenings. The following question is of interest when considering the efficiency of $A^*_\uparrow$ or A^*_{DW}: how do the methods behave in the case that a so-called *monotone* heuristic function guides the search? And: in this case, can there exist states that are *reopened* and expanded (again and again)? This question is answered in the paper, extending the scope of results given in [4]: here, the effect of relaxing the selection condition of A^* on potential reopenings of states has been considered.

This is of particular interest since the original A^*-algorithm is known to expand every distinct state *at most once* in the case of a monotone heuristic function [6]. If the methods for relaxed best-first search cannot guarantee the same, performance can be degraded. In a worst-case scenario, the overhead as caused by reopened states could even exceed the savings provided by the relaxation.

In this paper it is shown by examples that both discussed relaxation methods in fact can show the above (unwanted) behavior. As a remedy, new revised versions of the methods are suggested that expand each state at most once. The property of ϵ-admissibility must be reconsidered for the new approaches. Using the formal framework, first a general upper bound for the deviation of the solution from the optimum is derived. This upper bound is exponential in the depth of the search. Second, again by use of the framework, the bound can be tightened to an only *linear* bound in the case of a revised version of $A^*_{\uparrow}$. This result confirms a previous result for this special case, stated in a formal analysis of ARA^* [7]. As a benefit from the provided formal framework, our proof is kept considerably shorter and more concise, further strengthening the framework. Experimental results give a comparison of all methods with each other and with previous methods, showing the efficiency of the suggested approaches.

2 Search by A^*

A g- and an h-component associated with every state q are combined to the so-called evaluation function $\varphi(q) = g(q) + h(q)$. The minimal cost of a path from s to q is denoted $g^*(q)$. The minimal cost of a path from q to a goal state is denoted $h^*(q)$. To guarantee a minimal solution cost, it must be $h(q) \leq h^*(q)$ [6]. In this case, h is called *admissible*. A^* maintains a prioritized queue OPEN which is ordered with respect to increasing values $\varphi(q)$. In the beginning, this queue only contains the initial state s. At each step, a state q with a *minimal* φ-value is expanded, dequeued and put on a list called CLOSED. During expansion, the successor states of q are generated and inserted into the queue OPEN according to their φ-values. For this, the values g and h of the successor states are computed dynamically. The component g accumulates transition costs as the sum of the cost $c(r, r')$ of all transitions $r \longrightarrow r'$ occurring on the cheapest known path to q. If a path between q and q' is optimal, its cost is denoted by $k(q, q')$.

A successor state q' might be generated a second time if q' has more than one predecessor state. If a cheaper path from s to q' is found in this case, $g(q')$ is updated. If q was on the list CLOSED, q is reopened, i.e. it is put on OPEN again. By that, states get a second chance during the search for the minimum cost path when new information about them is available. The algorithm terminates if the next state to expand is a goal state t. The estimate $h(t) = h^*(t)$ must be zero. In this case, the path found up to t is of minimal cost, denoted C^*, and it is reported as solution.

A heuristic function h is said to be *monotone*, if $h(q) \leq k(q, q') + h(q')$ for all descendants q' of q. In [6] it is shown that, in the case of a monotone heuristic function h, A^* finds optimal paths to all expanded nodes. This ensures that every state is expanded at most once.

3 Previous Work

In this section previous work related to our approach is briefly reviewed.

3.1 Relaxing Best-First Search

To keep the paper self-contained, next a brief review of previous quasi-exact approaches based on A^* follows. All approaches relax some of the conditions used by A^* to derive a faster algorithm with a provable upper bound on sub-optimality. The ideas, how this is done, vary significantly and in the following three methods are distinguished.

Dynamic Weighting The idea of *Dynamic Weighting* (A^*_{DW}) [10] is to relax the fixed weighting of the breadth- and the depth-first component (i.e., of g and h) used by the additive evaluation function $\varphi(q) = g(q) + h(q)$ of A^*. Algorithm A^*_{DW} starts with a high weighting of the depth first component at the beginning of the search (as this may help to find a promising direction more quickly) and then dynamically weighs the depth-first component less heavily as the search goes deeper, preventing premature termination. For $\epsilon > 0$, the evaluation function used by A^*_{DW} is

$$\varphi^{\mathrm{DW}}(q) = g(q) + h(q) + \epsilon \cdot \left[1 - \frac{d(q)}{N}\right] \cdot h(q)$$

where $d(q)$ denotes the depth of the node representing state q in the search graph, and N denotes the depth of a goal node, respectively. Often, all paths to a node in this graph are of equal length, and thus this depth is the number of edges on such a path. If N is not known in advance, an upper bound or an estimate can be used instead.

It can be shown that A^*_{DW} is ϵ*-admissible*, i.e. it always finds a solution whose cost does not exceed the optimal cost by more than a factor of $1 + \epsilon$.

Constant Inflation More recently, a much simpler idea has been considered in [16] and also within the framework of a so-called *Anytime Repairing* A^*-algorithm (ARA^*) [7]: the *constant* inflation of the depth-first component by a fixed factor $1 + \epsilon$ ($\epsilon > 0$). That is, the evaluation function

$$\varphi^{\uparrow}(q) = g(q) + (1 + \epsilon) \cdot h(q)$$

is used instead of the original evalution function φ of A^*.

In comparison to A^*_{DW}, no other precautions against premature termination are taken here. However, it can be shown that bounding the inflation of h by the factor $1+\epsilon$ already suffices to guarantee the same bounded sub-optimality as with A^*_{DW}. This method is referred to as $A^*_{\uparrow}$ and it is further analyzed in Section 4.

Search Effort Estimates Experiments have shown the following: during execution of an A^*-algorithm, a large amount of time is spent discriminating among many paths whose cost do not vary significantly from each other. To assure optimality of the final solution, A^* spends a disproportionately long time to select the best of roughly equal candidate states as next state to expand. This behavior raises the idea of equipping A^* with the capability of terminating earlier with a sub-optimal but otherwise perfectly acceptable solution path.

In [9] an extension of A^* called A^*_ϵ has been proposed, that addresses the above problem by adding a second queue FOCAL which maintains a subset of the states on OPEN. This subset is the set of those states whose cost does not deviate from the minimum cost of a state on OPEN by a factor greater than $1 + \epsilon$. Formally,

$$\text{FOCAL} = \{q \mid \varphi(q) \leq (1+\epsilon) \cdot \min_{r \in \text{OPEN}} \varphi(r)\}). \tag{1}$$

The operation of A^*_ϵ is identical to that of A^* except that A^*_ϵ selects a state q from FOCAL with minimal value $h_F(q)$. The function h_F is a second heuristic estimating the computational effort required to complete the search. By this the nature of h_F differs significantly from that of h since h estimates the *solution cost* of the remaining path whereas h_F estimates the remaining *time* needed to find this solution. The choice of h_F puts a high degree of freedom to the approach which will be subject to further investigation in Section 4. In [9], it has been suggested to use

a) $h_F = h$ or
b) to integrate properties of the subgraph emanating from a given state q.

The motivation behind a) is that minimizing the h-component for the states in the set FOCAL means preferring the states with the highest g-component. Such states are least estimative and a fast completion of the best known path to such a state, i.e., a fast termination can be expected. As a concrete suggestion for b), $h_F(q) = N - d(q)$ will be used later in the experimental evaluation (see Section 7): to minimize $N - d(q)$ means to prefer the deeper states in the search graph. This is done with the motivation that the subgraphs emanating from them tend to be comparatively small and thus the same can be expected for the remaining run time. Also A^*_ϵ is *ϵ-admissible*, i.e. we have the upper bound $1 + \epsilon$ on sub-optimality.

In [4], an example was given that shows that Algorithm A^*_ϵ has a serious drawback: even when guided by a monotone heuristic h, A^*_ϵ can be doomed to *reopen* many states. This is a source of degradation in run time and contrasts to the behavior of classical A^* which expands every state at most once [6].

The following remedy has been suggested in [4]: instead of maintaining closed states on a list CLOSED, states are simply marked as closed after expansion and removal from OPEN. If the method finds a better path for a state q marked as closed, *this better path is ignored*, i.e. $g(q)$ is *not* updated. Otherwise, method A^{pprox} follows the usual operation of A^*_ϵ. Although ϵ-admissibility can not be guaranteed for A^{pprox} in general, still the following result holds [4].

Theorem 1. *Let N be the maximal length of a solution path. When driven by a monotone heuristic, algorithm A^{pprox} always finds a solution not exceeding the optimal cost by a factor greater than $(1+\epsilon)^{\lfloor \frac{N}{2} \rfloor}$.*

For smaller values of ϵ, this bound still is useful. Note that during practical operation, A^{pprox} usually is far off this worst-case, i.e. it may yield much better results.

4 Unifying View

In this section, a framework provides a unifying view of the three approaches of the previous section. They are characterized as special instances of one generic relaxation algorithm. As the first step, the next result states a condition that guarantees the conformity of an evaluation function with the strategy described in Section 3.1.

Theorem 2. *Let us consider a state space together with an evaluation function $\varphi = g + h$ and let* FOCAL *be defined as before in Equation (1). For all states q of the state space, let $\varphi^{\Uparrow}(q) = (1+\epsilon) \cdot \varphi(q)$ and let $\varphi(q) \leq \varphi'(q) \leq \varphi^{\Uparrow}(q)$. Let*

$$\hat{q} = \arg \min_{q \in \text{OPEN}} \varphi'(q).^{1}$$

Then it must be that $\hat{q} \in$ FOCAL.

Proof. See the Appendix.

In Section 3.1 it has been mentioned that the choice of the heuristic function h_F which estimates the remaining search effort leaves a considerable degree of freedom to the method. Next we go one step further, clarifying that the discussed relaxation methods can be *rediscovered* simply by respective choices for h_F. In detail, Theorem 2 allows to characterize A^*_{DW} and $A^*_{\uparrow}$ as two instantiations of the generic method given in Section 3.1. In this, Pearl and Kims' proposal proves to be more than just another relaxation algorithm: the next result shows that it also serves as a framework for the relaxation of best-first search in general.

Theorem 3. *Let us consider the graph representation of a state space and let g, h be the breadth-first and the depth-first component of a relaxed best-first search algorithm, respectively. For all states q of the state space, let $\varphi^{\uparrow}(q) = g(q) + (1+\epsilon) \cdot h(q)$, let $d(q)$ denote the depth of the node representing q, let N denote the depth of a goal node, and let $\varphi^{\text{DW}} = g(q) + h(q) + \epsilon \cdot \left[1 - \frac{d(q)}{N}\right] \cdot h(q)$. Further, assume that identical tie-breaking rules are used in the algorithms. Then we have:*

- *The operation of Algorithm A^*_{DW} is identical to that of A^*_{ϵ} with search estimate $h_F = \varphi^{\text{DW}}$, and*
- *The operation of Algorithm $A^*_{\uparrow}$ is identical to that of A^*_{ϵ} with search estimate $h_F = \varphi^{\uparrow}$.*

Proof. See the Appendix.

In brief, the result states that the choice of the next node to expand as performed by A^*_{DW} and $A^*_{\uparrow}$ conforms to the relaxation strategy of A^*_{ϵ} as stated in Equation (1). Notice that, despite the fact that A^*_{DW} and $A^*_{\uparrow}$ are formulated by use of evaluation functions that are different from that of A^* or A^*_{ϵ} (i.e., different from $\varphi = g + h$), they provably

[1] Throughout the paper, the following notation is used: $\arg\min_{x \in S} f(x)$ returns one $x \in S$ that minimizes the function f.

act as if $\varphi = g + h$ would be used. This holds since they are also guided by the second heuristic h_F. It is precisely this function h_F that then must be replaced by the respective alternative evaluation function.

From now on, it will be distinguished between the new *framework* provided by A^*_ϵ with this result and *instantiations* as one particular algorithm, e.g. as the algorithm proposed in [9]. Hence, it is denoted A^*_ϵ, $h_F = \ldots$ whenever a particular algorithm is addressed, the introduced framework itself however is referred to as A^*_ϵ, i.e. without giving a particular second heuristic function h_F.

The result of Theorem 3 allows to transfer any provable result for A^*_ϵ directly to A^*_{DW} and $A^*_\uparrow$, as the two methods are special instances of A^*_ϵ. This results in the following theorem which considers under what conditions states are eligible for state expansion. It generalizes a known result in [9] and is helpful for efficiency considerations and comparisons.

Theorem 4. *Consider a state space with cost function g. Let us assume an admissible heuristic function h, and consider an optimal path $s, \ldots, q'$ where q' is the first state that currently also appears on* Open *during a (relaxed) best-first search algorithm A. Further, let $\varphi = g + h$.*

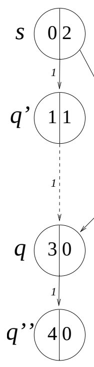
– $A = A^*$*: $\varphi(q) \leq C^*$ for all states expanded.*
– $A = A^*_\epsilon$*: $\varphi(q) \leq (1+\epsilon) \cdot C^*$ for all states expanded.*
– $A = A^*_\uparrow$*: for all states expanded, either $\varphi(q) \leq C^*$ holds or we have $\varphi(q) > C^*$* and $\varphi(q) \leq \mathrm{UB}$ *where for $h(q) \in [0, h(q')[$,* UB *ranges from $(1+\epsilon) \cdot C^*$ to C^*, not including C^*.*
– $A = A^*_{\mathrm{DW}}$*: for all states expanded, either $\varphi(q) \leq C^*$ holds or we have $\varphi(q) > C^*$* and $\varphi(q) \leq \mathrm{UB}$ *where for $h(q) \in \left[0, \frac{N-d(q')}{N-d(q)} \cdot h(q')\right[$,* UB *ranges from $(1+\epsilon) \cdot C^*$ to C^*, not including C^*.*

Proof. See the Appendix.

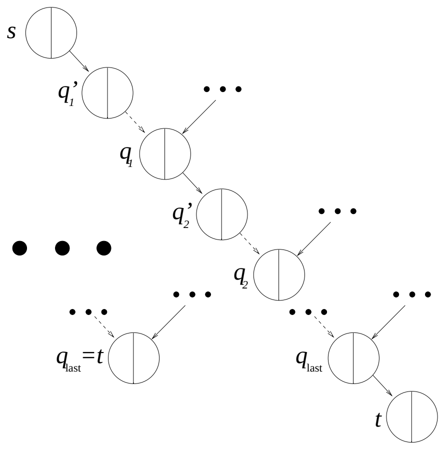
This result indicates the following: Both for $A^*_\uparrow$ and for A^*_{DW}, states q with $C^* < \varphi(q) \leq (1+\epsilon) \cdot C^*$ can only be eligible to expansion if their φ-value also stays below the stated upper bound UB. To achieve the value $(1+\epsilon) \cdot C^*$ for UB, the h-value of the eligible state must be much less than $h(q')$ and/or the eligible state must reside at a significantly deeper level in the search graph.

This contrasts to the situation in A^*_ϵ where no such additional restriction holds for the eligibility for expansion. Moreover, in many AI problem domains typically a lot more of breadth-fist search than depth-first search will be performed by an exact or quasi-exact best-first search method (this also holds for BDD minimization). As a consequence, during a typical algorithm run, often states with equal or similar h-values and/or depth are expanded in a series of consecutive expansions. Thus, eligible states that really are far enough “below” q' (in terms of the h-value and/or depth) to be chosen for expansion, are rare. Hence, for an eligible state q, $\varphi(q) \leq C^*$ often is much more typical.

Consequently it can be expected that the total number of states expanded during run of $A^*_\uparrow$ and A^*_{DW} typically is at least no more than that for A^* (and often much less). This



(a) An example for a sub-optimal path to an expanded state.



(b) Worst-case scenario with $\lfloor \frac{N}{2} \rfloor$ deviations, even and odd case.

Fig. 1. Examples for the behavior with and without reopenings.

number is expected to still remain this low in the situations where A^*_ϵ using $h_F = h$ or $h_F = N - d(q)$ runs into problems.

5 Monotonicity

In Section 1 the following question has been raised: provided that Algorithm $A = A^*_\uparrow, A^*_{\text{DW}}$ is guided by a monotone heuristic h, can states be *reopened*?

Next we give an example which shows that such states may exist for both choices of A. In Fig. 1(a), the left datum annotated at a node is the g-value, the right one is the h-value. Edges depict state transitions and the cost of the transition is annotated at each edge. The heuristic function h is monotone since the series of φ-values is monotonic non-decreasing along every path in the state space graph. In the case $A = A^*_\uparrow$, let $\epsilon = \frac{3}{2}$. In the case $A = A^*_{\text{DW}}$, let the anticipated depth of a goal state $N = 3$ and $\epsilon = \frac{9}{4}$. It is easily verified that q is reopened for these choices.

To further analyze the operation of $A^*_\uparrow$, the following new result states an upper bound for the deviation of g from g^* for an expanded state.

Lemma 1. *Let $\epsilon > 0$. The paths to expanded states found by an $A^*_\uparrow$-algorithm that is guided by a monotone heuristic may be sub-optimal. However, this deviation is* bounded, *in detail:*

$$\forall q \in \textsc{Closed} : g(q) - g^*(q) \leq \epsilon \cdot k(q', q) \tag{2}$$

where q' is the first state on Open *on an optimal path $s, \ldots, q', \ldots, q$ at the time of expansion of q.*

Proof. See the Appendix.

6 Preventing to Reopen States

The problem discussed in Section 5 can be addressed by the same strategy as for A^{pprox} (see Section 3). The respective approaches that do not reopen any state will be called $A^{pprox}_{\uparrow}$ and A^{pprox}_{DW}, respectively. As a consequence of Theorem 3, the exponential upper bound of Theorem 1 for A^{pprox} also holds for $A^{pprox}_{\uparrow}$ and A^{pprox}_{DW}. However, in the case of $A^{pprox}_{\uparrow}$, the upper bound can be strongly tightened:

Theorem 5. *When driven by a monotone heuristic, algorithm $A^{pprox}_{\uparrow}$ always finds a solution not exceeding the optimal cost by a factor greater than $1 + \epsilon$.*

Proof. See the Appendix.

Using the introduced framework, the proof is considerably more concise than that of a similar previous result in [7]. Basically, the proof follows a similar flow of arguments as the proof for ϵ-admissibility of A^*_ϵ [9], except that, due to the modified behavior of the algorithm, we have to account for the following consequence. In $A^{pprox}_{\uparrow}$, the g-value of states on an optimum path may irrecoverably be affected by deviations from the optimum g^*: by Lemma 1, states might be expanded while the best known path to them is still sub-optimal and, due to the modified behavior of $A^{pprox}_{\uparrow}$, no reopening/improvement can take place later. This effect increases the maximum deviation on an optimal path. To what extent, is determined in the worst-case scenario: let N be the maximal length of a path. Since always *two* nodes must be involved for a deviation of a g-value to occur (see the proof of Lemma 1), the deviation of a g-value from g^* increases at most $\lfloor \frac{N}{2} \rfloor$ times, see Fig. 1(b): dashed transition are “late” transitions, i.e. the state they lead to has already been opened along a sub-optimal path different from p. State q_{last} is the last state that has been prematurely opened along such a sideway and thus is affected by a deviation of the g-value. We have $q_{\mathrm{last}} = q_{\lfloor \frac{N}{2} \rfloor}$, regardless whether n is odd or even (see Fig. 1(b)). The proof then is an induction on $i = 1, \ldots, \lfloor \frac{N}{2} \rfloor$.

7 Experimental Results

To evaluate the algorithms described by our framework, respective methods targeting the quasi-exact minimization of reduced ordered *Binary Decision Diagrams* (BDDs) have been implemented. BDDs were introduced in [2] and are well known from hardware verification and logic synthesis. They are *Directed Acyclic Graphs* (DAGs) representing Boolean functions where a Shannon decomposition in a Boolean variable is carried out with each node. Reduced diagrams are considered, derived by removing redundant nodes and merging isomorphic subgraphs. For more details see [2].

Heuristic BDD minimization is done by thumb rules to reorder the Boolean variables, e.g. [11]. The results are often far away from the optimum. For some applications, this is a significant drawback. Especially in applications like logic synthesis targeting multiplexor design styles, e.g. [8, 14], it is important to determine a good ordering, since a reduction in the number of BDD nodes directly transfers to a smaller chip area. For this reason, there is a high demand for faster exact or approximate methods with bounded sub-optimality.

It has been shown that it is NP-complete to decide whether the number of nodes of a given BDD can be improved by variable reordering [1]. Moreover, the existence of a polynomial algorithm to approximate the optimal variable ordering of BDDs implies $P = NP$ [12]. For this reason, as with exact methods, the run time of an approximate method to improve the variable ordering is expected to be much higher than that of heuristics.

All experimental results have been carried out on a machine with a Dual Xeon processor running at 3.2 GHz, with a main memory of 12 GByte and a run time limit of 3,600 CPU seconds. The memory requirement of all evaluated methods never exceeds 500 MBytes, hence no memory limit had to be applied. Three previous methods have been implemented: the first is called A^{pprox} as described in [4], the second is called Dynamic Weighting (A^*_{DW}) [10]. An idea of [16] and of the so-called ARA^* algorithm [7], namely the constant inflation of the heuristic function as described in Section 3.1, has been implemented as the approach $A^*_{\uparrow}$. Moreover, revised versions of the mentioned methods have been implemented as the corresponding methods A^{pprox}_{DW}, a revised version of A^*_{DW}, and as the method $A^{pprox}_{\uparrow}$, a revised version of $A^*_{\uparrow}$. To put up a testing environment, all algorithms have been integrated into the CUDD package [13]. By this it is guaranteed that they run in the same system environment. In the experiments, the methods have been applied to BDDs built from a set of standard benchmark circuits of LGSynth93 [3]. The implementation of all algorithms is based on the implementation of the A^*-based approach to exact BDD minimization of [5].

In a first series of experiments A^* and A^{pprox} have been compared to $A^{pprox}_{\uparrow}$. The results are depicted in Fig. 2. In contrast to the behavior of A^{pprox}, the run time of $A^{pprox}_{\uparrow}$ is monotonically decreasing. This confirms the result of Theorem 4 and shows that also the revised version of $A^*_{\uparrow}$, i.e. $A^{pprox}_{\uparrow}$, behaves according to the upper bounds stated in Theorem 4. For $A^{pprox}_{\uparrow}$, the degradation of solution quality first increases slowly (e.g., for $\epsilon \in [0, 0.5]$) and later ascends more steeply with increasing ϵ. When comparing the run time of $A^{pprox}_{\uparrow}$ to that of A^*, Fig. 3 illustrates how the gain in run time grows monotonically with the degree of relaxation (the curve in the space spanned by the percentual gain and the degree ϵ is a convex hyperbola). At the higher relaxation degree of $\epsilon = 3$ the reduction in run time is already more than 90% on average. Taking into account that $A^{pprox}_{\uparrow}$ also has much more convenient theoretical properties than A^{pprox} (in particular, $A^{pprox}_{\uparrow}$ guarantees a much tighter upper bound for the deviation of the solution from the optimum), $A^{pprox}_{\uparrow}$ proves to be clearly superior to A^{pprox} both from a theoretical and a practical standpoint. As Fig. 4 shows, high speed-ups can be obtained at an only small degradation of solution quality. In fact the average degradation is considerably much less than the worst-case degradation by a factor of $1 + \epsilon$ as guaranteed by the theory. Operating at 40% of relaxation, on average the results are only 0.5% larger than the optimum BDD size. When using a degree of relaxation of 100%, i.e. when theoretically allowing for solutions that are twice the minimum size, the average degradation still is only 4.3%. Motivated by these positive results, also very high relaxations have been examined: Fig. 4 shows that the average degradation stays below 20% for a wide range of high relaxation degrees, it first reaches 20.5% for $\epsilon = 20$. Moreover, the resulting plot forms a convex hyperbola where the steepness decreases with ascending degree of relaxation. In a second series of experiments, $A_{\uparrow}^{pprox}$ has been compared to A_{DW}^{pprox} in terms of quality and run time. Due to space limitation, the results of these experiments have not been included. Summarized, A_{DW}^{pprox} has significantly higher run times than $A_{\uparrow}^{pprox}$ (20-30%) while at the same time slightly better results can be obtained, i.e. there is a significant penalty for small improvements in quality (below 3%) provided by A_{DW}^{pprox}.

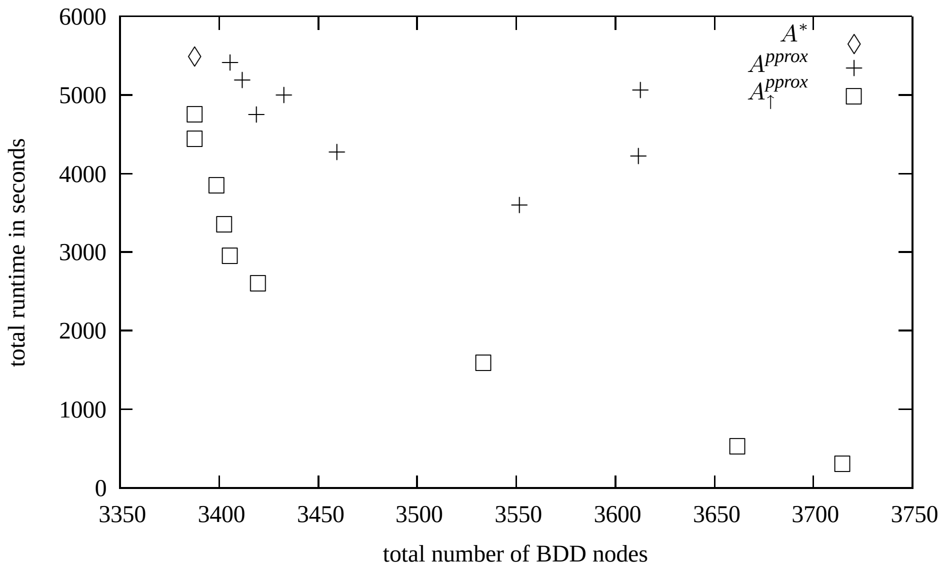


Fig. 2. Trading off run time for solution quality with $\boldsymbol{A}^{pprox}$ and $\boldsymbol{A}_{\uparrow}^{pprox}$.

8 Conclusions

A new framework for previous and new extensions of the A^*-algorithm has been presented. It describes a class of generic algorithms that tolerate provably bounded worst-case increases in solution cost. This is achieved by different ideas to relax some of the conditions of A^*, and happens in favour of smaller search efforts required to complete the algorithm. The user has full control of the degree of relaxation and can trade off run time for quality of the solution.

Besides the formal contributions of the paper, two new methods are derived from the framework. They guarantee to expand every state at most once if provided with a so-called monotone heuristic. This can largely reduce the run time and also strengthens the robustness of the approaches.

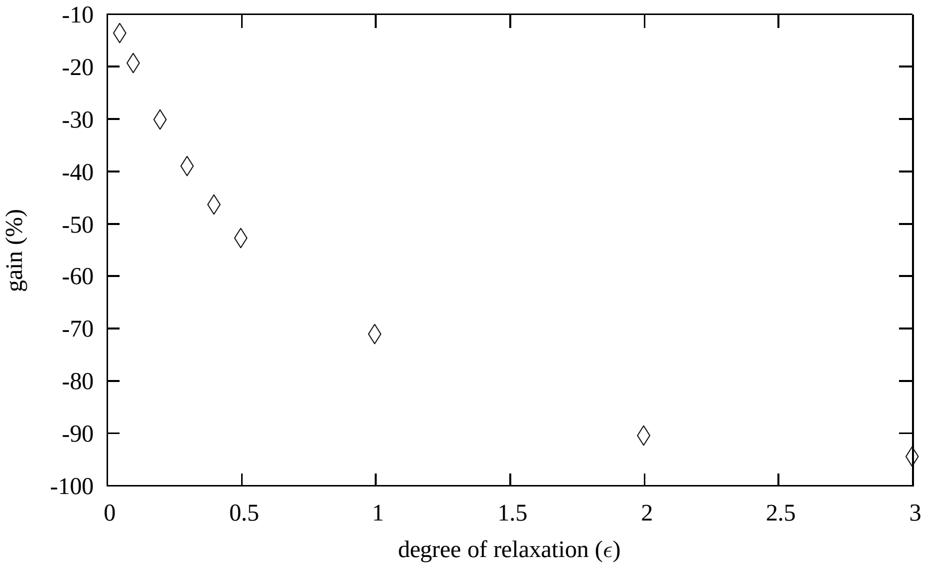


Fig. 3. Degree of relaxation vs. gain in run time of $A_{\uparrow}^{pprox}$.

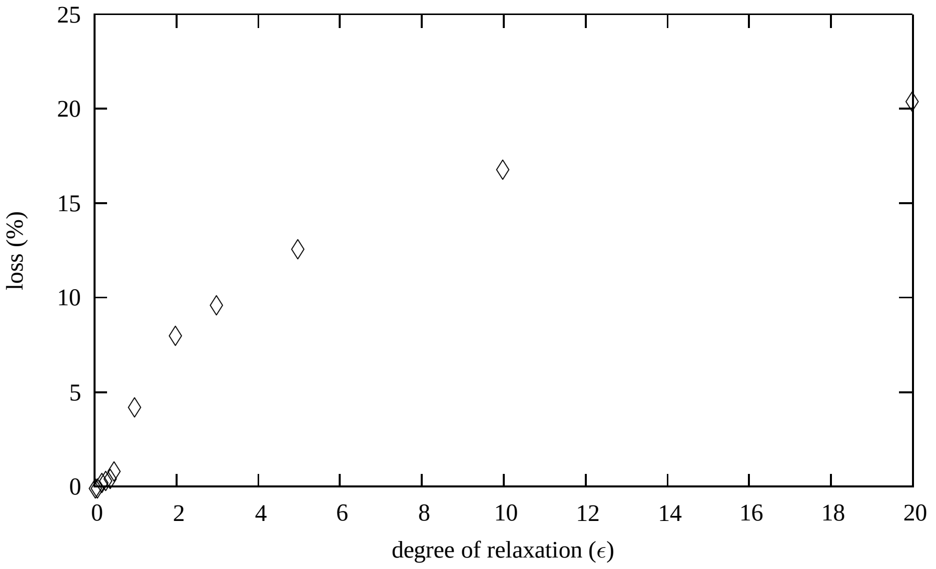


Fig. 4. Degree of relaxation vs. loss in quality of $A_{\uparrow}^{pprox}$.

Experimental results are reported that clearly demonstrate the efficiency of the presented new approaches. A comparison to the best known exact BDD minimization algorithm (which is based on the generic A^* algorithm) and to a previous relaxed method shows reductions in run time by up to one order of magnitude. This is obtained while the degradation of solution quality is provably bounded and stays below a few percents on average.

Appendix

Proof of Theorem 2. Let $q_0 = \arg\min_{q \in \text{OPEN}} \varphi(q)$. We have

$$\begin{aligned}
\varphi(\hat{q}) &\leq \varphi'(\hat{q}) && (3)\\
&\leq \varphi'(q_0) && (4)\\
&\leq \varphi^{\Uparrow}(q_0) && (5)\\
&= (1+\epsilon)\cdot\varphi(q_0) && \\
&= (1+\epsilon)\cdot \min_{q\in\text{OPEN}} \varphi(q). && (6)
\end{aligned}$$

Equation (3) holds by the definition of φ' in the assumption. Next, Equation (4) holds by definition of $\hat{q}$. Then, Equation (5) holds again with the definition of φ'. By Equation (6), $\hat{q} \in$ FOCAL already follows. □

Proof of Theorem 3. First it is easily verified that $\varphi(q) \leq \varphi^{DW}(q) \leq \varphi^{\uparrow}(q) \leq \varphi^{\Uparrow}(q)$ for all states q of the considered state space. By Theorem 2, the respective next state expanded by $A^*_{\uparrow}$ and A^*_{DW} must be contained in FOCAL. Second, A^*_{ϵ} chooses a state q_F from FOCAL with $q_F = \arg\min_{q\in\text{FOCAL}} h_F(q)$. As h_F is assigned to the respective evaluation function, and since the same respective tie-breaking rule is used, A^*_{ϵ} must act exactly as A^*_{DW} and $A^*_{\uparrow}$, respectively. □

Proof of Theorem 4. The results for the cases $A = A^*, A^*_{\epsilon}$ are already well-known [6, 9]. They are merely opposed here to the new results. Because q is expanded before q',

$$\varphi^{\uparrow}(q) = \varphi(q) + \epsilon\cdot h(q) \leq \varphi^{\uparrow}(q') \tag{7}$$

in the case $A = A^{\uparrow}$, and

$$\varphi^{\text{DW}}(q) = \varphi(q) + \epsilon\cdot\left[1 - \frac{d(q)}{N}\right]\cdot h(q) \leq \varphi^{\text{DW}}(q') \tag{8}$$

in the case of $A = A^{\text{DW}}$. To derive the stated upper bounds for $A = A^{\uparrow}, A^{\text{DW}}$, it now suffices to separate $\varphi(q)$ on the left side of the two equations (7) and (8), respectively. The upper bounds range within the stated intervals since

– the term $h(q')$ can be bounded by $h^*(q')$ because of the admissibility of h, and
– since an optimal path is considered, we have $g(q') = g^*(q')$ and finally $\varphi^*(q') \leq C^*$.

□

Proof of Lemma 1. Consider an optimal path p from s to q. Let q' be the first state on $p = s, \ldots, q', \ldots, q$ which also appears on OPEN[2]. Assume that $q \neq q'$ and assume that q is selected for expansion. Since q is expanded before q', we have $\varphi^{\uparrow}(q) \leq \varphi^{\uparrow}(q')$, and, with the optimality of p and the monotonicity of h

$$\begin{aligned} g(q) + (1+\epsilon) \cdot h(q) &\leq g(q') + (1+\epsilon) \cdot h(q') \\ &\leq g(q') + (1+\epsilon) \cdot (k(q', q) + h(q)) \\ &= g^*(q') + k(q', q) + \epsilon \cdot k(q', q) + (1+\epsilon) \cdot h(q) \\ &= g^*(q) + \epsilon \cdot k(q', q) + (1+\epsilon) \cdot h(q). \end{aligned}$$

Hence, $g(q) \leq g^*(q) + \epsilon \cdot k(q', q)$ and Equation (2) follows. □

Proof of Theorem 5. Let N be the maximal length of a search path. The proof uses

$$g(q_i) \leq g(q_i') + (1+\epsilon) \cdot k(q_i', q_i) \text{ for } 1 \leq i \leq \left\lfloor \frac{N}{2} \right\rfloor. \tag{9}$$

This follows from the admissibility of h and the fact that q_i is expanded before q_i': the latter implies $\varphi^{\uparrow}(q_i) \leq \varphi^{\uparrow}(q_i')$, thus $g(q_i) + (1+\epsilon) \cdot h(q_i) \leq g(q_i') + (1+\epsilon) \cdot h(q_i') \leq g(q_i') + (1+\epsilon) \cdot [k(q_i', q_i) + h(q_i)]$ and Equation (9) follows. We show

$$g(q_i) - q^*(q_i) \leq \epsilon \cdot \sum_{l=1}^{i} k(q_l', q_l) \text{ for } 1 \leq i \leq \left\lfloor \frac{N}{2} \right\rfloor \tag{10}$$

by induction on i. Then, by the optimality of the considered path, the deviation of $g(q_{\text{last}}) = g(q_{\lfloor \frac{N}{2} \rfloor})$ must be less or equal than $\epsilon \cdot C^*$ and finally also the deviation of the computed solution from the optimum can not be greater than $1 + \epsilon$.

To start the induction, let $i = 1$: the claim of Equation (10) holds by Lemma 1. The lemma can be applied here since the operation of $A^*_{\uparrow}$ and $A^{pprox}_{\uparrow}$ is identical before the first state has been reopened. Now let us assume that Equation (10) holds for i. For the step $i \longrightarrow i+1$ we derive

$$\begin{aligned} g(q_{i+1}) - g^*(q_{i+1}) &\leq g(q_{i+1}') + (1+\epsilon) \cdot k(q_{i+1}', q_{i+1}) - g^*(q_{i+1}) && (11) \\ &= k(q_i, q_{i+1}') + (1+\epsilon) \cdot k(q_{i+1}', q_{i+1}) + g(q_i) - g^*(q_{i+1}) && (12) \\ &= \epsilon \cdot k(q_{i+1}', q_{i+1}) + g(q_i) - g^*(q_i) && (13) \\ &= \epsilon \cdot k(q_{i+1}', q_{i+1}) + \epsilon \cdot \sum_{l=1}^{i} k(q_l', q_l) && (14) \\ &= \epsilon \sum_{l=1}^{i+1} k(q_l', q_l). \end{aligned}$$

[2] Note that it is straightforward to prove that, during operation of the algorithm, at least one state on p must be an open state. The proof is an induction on the length of p which is started by s, the very first state occuring both on OPEN and p.

Equation (11) holds by Equation (9). Since q'_{i+1} is traversed via q_i on an optimal path, $g(q'_{i+1}) = k(q_i, q'_{i+1}) + g(q_i)$. Thus, Equation (12) follows. We have $g^*(q_{i+1}) = g^*(q_i) + k(q_i, q'_{i+1}) + k(q'_{i+1}, q_{i+1})$ since an optimal path is considered. Thus, Equation (13) follows. Using the induction hypothesis, i.e. Equation (10), next Equation (14) is obtained, completing the proof. □

References

1. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs in NP-complete. *IEEE Trans. on Comp.*, 45(9):993–1002, 1996.
2. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.
3. Collaborative Benchmarking Laboratory. *1993 LGSynth Benchmarks*. North Carolina State University, Department of Computer Science, 1993.
4. R. Ebendt and R. Drechsler. Quasi-exact BDD minimization using relaxed best-first search. In *IEEE Annual Symp. on VLSI*, pages 59–64, 2005.
5. R. Ebendt, W. Günther, and R. Drechsler. Combining ordered-best first search with branch and bound for exact BDD minimization. *IEEE Trans. on CAD*, 24(10):1515–1529, 2005.
6. P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 2:100–107, 1968.
7. M. Likhachev, G. Gordon, and S. Thrun. ARA^*: Formal analysis. Technical report of the Carnegie Mellon University, 2003.
8. L. Macchiarulo, L. Benini, and E. Macii. On-the-fly layout generation for PTL macrocells. In *Design, Automation and Test in Europe*, pages 546–551, 2001.
9. J. Pearl and J. Kim. Studies in semi-admissible heuristics. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-4(4):392–399, 1982.
10. I. Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proc. 3rd Int. Joint Conf. on Artifi cial Intelligence.*, pages 12–17, 1973.
11. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.
12. D. Sieling. Nonapproximability of OBDD minimization. *Information and Computation*, 172(2):103–138, 2002.
13. F. Somenzi. *CU Decision Diagram Package Release 2.4.0*. University of Colorado at Boulder, 2004.
14. C. Yang and M. Ciesielski. BDS: a BDD-based logic optimization system. *IEEE Trans. on CAD*, 21(7):866–876, 2002.
15. R. Zhou and E. Hansen. Memory-bounded A^* graph search. In *15th Int. Florida Artifi cial Intelligence Research Soc. Conf.*, pages 203–209, 2002.
16. R. Zhou and E. Hansen. Multiple sequence alignment using A^*. In *Proc. of the National Conference on Artifi cial Intelligence, Student Abstract*, 2002.