Shortest Path Planning for Energy-Constrained Mobile Platforms Navigating on Uneven Terrains

Nuwan Ganganath, Chi-Tsun Cheng, Tyrone Fernando, Herbert H. C. Iu, and and Chi K. Tse

***Abstract*—Finding a shortest feasible path between two given locations is a common problem in many real-world applications. Previous studies have shown that mobile platforms would consume excessive energy when moving along shortest paths on uneven terrains which often consist of rapid elevation changes. Mobile platforms powered by portable energy sources may fail to follow such paths due to the limited energy available. This paper proposes a new heuristic search algorithm called *Constraints Satisfying A* (CSA*)* to find solutions to such resource constrained shortest path problems. When CSA* is guided by admissible heuristics, it guarantees to find a globally optimal solution to a given constrained search problem if such a solution exists. When CSA* is guided by consistent heuristics, it is optimally efficient over a class of equally informed admissible constrained search algorithms with respect to the set of paths expanded. Test results obtained using real terrain data verify the applicability of the proposed algorithm in shortest path planning for energy-constrained mobile platforms on uneven terrains.**

***Index Terms*—CSA*, multiple resource constraints, shortest paths, heuristic search, outdoor navigation.**

I. Introduction

PATH planning is a process of finding a desired path from a set of possible paths between two given locations such that some predefined requirements are satisfied. It has been a topic of interest for many decades with an early focus on finding least cost paths between nodes in weighted graphs [1]–[3]. Recently, path panning in outdoor environments has drawn a significant attention with the emergence of autonomous cars [4]–[6], mobile sensor networks [7]–[9], and planetary rovers [10], [11]. Outdoor environments often consist of uneven terrains on which some paths are not physically feasible for mobile platforms due to their instability on steep slops and motion power limitations [12]–[14]. Many existing outdoor path planning algorithms focus either on finding shortest paths [15]–[17] or energy-optimal paths [18]–[23].

This work is supported by the Hong Kong PhD Fellowship Scheme, the Endeavour Research Fellowship from the Australian Government, and the Department of Electronic and Information Engineering, the Hong Kong Polytechnic University (Project G-YBXK).

N. Ganganath, T. Fernando, and H.H.C. Iu are with the School of Electrical, Electronic and Computer Engineering, the University of Western Australia, Crawley, WA, Australia (email: nuwan@ganganath.lk; tyrone.fernando@uwa.edu.au; herbert.iu@uwa.edu.au).

C.-T. Cheng and Chi K. Tse are with the Dept. of Electronic and Information Engineering, the Hong Kong Polytechnic University, Hung Hom, Koloown, Hong Kong (email: ctcheng@ieee.org; michael.tse@polyu.edu.hk).

A part of this work was conducted when N. Ganganath was with the Dept. of Electronic and Information Engineering, the Hong Kong Polytechnic University, Hung Hom, Koloown, Hong Kong.

A. Background

Anisotropic friction and gravity effects have to be addressed when planning paths on uneven terrains. A physical model which can be used to calculate energy-cost of mobile platforms navigating under such external forces was proposed in [19]. The proposed model has also considered impermissible traversal headings due to power limitations and overturn dangers. An A* search algorithm [2] together with the energy-cost model and polyhedral terrain models have been used to find energy-efficient paths on uneven terrains. This energy-cost model has later been used for finding near-optimal energy paths on the surface of a vertical-axis ideal cone with anisotropic friction and gravity effects [20]. It has been shown that cone surface patches result in better terrain models and near-optimal paths obtained on such models are not much more complex than those on polyhedral terrain models. A terrain face weight concept was introduced in [15] to find shortest anisotropic paths on uneven terrains. Face weights capture some location based parameters of the terrain such as friction and slope. A polynomial time approximation algorithm was also proposed in [15] for finding shortest anisotropic paths.

Increased use of battery-powered mobile platforms has stimulated further research in energy-efficient path planning problems. In [23], energy-efficient path planning on steep terrains where mobile platforms can only move downhill has been considered. Lower- and upper-bound results on the combinatorial size of optimal paths and an approximation algorithm for finding energy-efficient paths are proposed in the same paper. Recently, a heuristic search algorithm called Z* was proposed for finding energy-efficient paths on uneven terrains [18]. It uses a heuristic function which can estimate heuristic energy-cost on uneven terrains using zigzag-like path patterns. It has been proven that Z* is capable of finding energy-optimal paths on a given terrain if such paths exist. Some other research focus on energy-efficient and shortest path replanning in dynamic and unknown outdoor environments [12], [16], [17], [22].

On uneven terrains, shortest paths are not always energy-efficient for mobile platforms as they have to often deal with rapid elevation changes. On the other hand, energy-optimal paths can be considerably longer than the shortest paths as they tend to follow equipotential curves, thus, result in extended journey times. Multiobjective search algorithms have been adapted in recent research [24] to avoid this trade-off between shortest and energy-optimal paths by finding a set of all nondominated paths between two given locations on uneven terrains. Finding such a set of paths, however, is computationally expensive. Therefore, it is essential to introduce more computationally efficient algorithms for finding shortest feasible paths for energy-constrained mobile platforms operating in outdoor environments.

In this work, the problem of finding energy-constrained shortest paths is considered as a resource constrained shortest path problem. It is a combinatorial optimization problem which can be defined on a digraph where a feasible optimal path between two given nodes need to be identified subjected to some given constraints. A graph search algorithm is *admissible* if it guarantees to find an optimal solution for any given problem if such a solution exists [3]. The focus of this paper is on finding such an admissible search algorithm.

B. Contributions and Organization of the Paper

This paper proposes an A*-like heuristic search algorithm, called *Constraints Satisfying A** (CSA*), to find an optimal solution to a given constrained search problem using a best-first search strategy. The nature of heuristics and the number of constraints govern the behavior of the search process of CSA* as well as the properties of its search results. Theoretical analyses are provided on the admissibility and efficiency of the proposed CSA*. It has been shown that CSA* finds an optimal solution subjected to multiple constraints when its search is guided by admissible heuristics. The search procedure of CSA* avoids unnecessary operations by performing a goal directed search. The computational efficiency of search algorithms that operate on graphs with multiple edge costs (edge cost vectors) are justified based on the number of path expansions triggered [25]. When guided by consistent heuristics, CSA* is proven to be optimally efficient over other equally informed admissible constrained search algorithms which use path selection and expansion as their basic operations. Test results obtained on sections of Matheny ridge and Anderson canyon verify the applicability of CSA* in shortest path planning for energy-constrained mobile platforms on uneven terrains.

The rest of the paper is organized as follows. Some preliminaries and the problem definition are respectively given in Sections II and III. The proposed CSA* algorithm is introduced and illustrated with a worked example in Section IV. Test results obtained using real terrain data are discussed in Section V. Basic properties of CSA*, including its strengths and limitations, are discussed in Section VI. Conclusions are given in Section VII. Finally, rigorous theoretical analyses on both the admissibility and efficiency of CSA* are provided in appendices.

II. Preliminaries

A. Dominance

Dominance can be identified as a partial order preference relation that is used in multiobjective problems [25]. For two real vectors $\vec{a} = [a_0, a_1, \dots, a_m]$ and $\vec{b} = [b_0, b_1, \dots, b_m]$, the relationship $\vec{a}$ *dominates* $\vec{b}$ is denoted by $\vec{a} \prec \vec{b}$ and defined as

$$\vec{a} \prec \vec{b} \quad \Leftrightarrow \quad \forall i \in [0, m] \mid a_i \leq b_i \wedge \vec{a} \neq \vec{b}.$$

A weaker version of the above relationship, $\vec{a}$ *dominates or equals* $\vec{b}$, can be denoted by $\vec{a} \preceq \vec{b}$ and defined as

$$\vec{a} \preceq \vec{b} \quad \Leftrightarrow \quad \forall i \in [0, m] \mid a_i \leq b_i.$$

A violation of the later relationship can be denoted by $\vec{a} \npreceq \vec{b}$ and defined as

$$\vec{a} \npreceq \vec{b} \quad \Leftrightarrow \quad \exists i \in [0, m] \mid a_i > b_i.$$

For example, let $\vec{x} = [1, 5]$, $\vec{y} = [2, 3]$, and $\vec{z} = [4, 5]$. Then following relationships hold: $\vec{x} \prec \vec{z}$, $\vec{x} \preceq \vec{z}$, $\vec{y} \prec \vec{z}$, $\vec{y} \preceq \vec{z}$, $\vec{y} \npreceq \vec{x}$, and $\vec{x} \npreceq \vec{y}$.

B. Paths and Path Costs

Consider a finite digraph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ that consists of $|\mathcal{N}|$ number of nodes and $|\mathcal{E}|$ number of edges. A search algorithm operating on $\mathcal{G}$ has to take only a finite number of decisions in each step. Let $\vec{c}(n, n') = [c_0(n, n'), c_1(n, n'), \dots, c_m(n, n')]$ be a non-negative cost vector which associates with every ordered pair of nodes $(n, n') \in \mathcal{E}$. A traversal between two nodes results in $(m + 1)$ distinct costs.

Let node n_k be accessible from node n_i. A path going from n_i to n_k is denoted by $\lambda_{n_i n_k}$, which is a sequence of nodes in $\mathcal{N}$ such that $(n_j, n_{j+1}) \in \mathcal{E}$ for all $i \leq j < k$. A set of all such paths from n_i to n_k is denoted by $\Lambda_{n_i n_k}$, thus, $\lambda_{n_i n_k} \in \Lambda_{n_i n_k}$. The cost vector of $\lambda_{n_i n_k}$ can be calculated as

$$\vec{c}(\lambda_{n_i n_k}) = \sum_{j=i}^{k-1} \vec{c}(n_j, n_{j+1}).$$

A scalar quantity $k_l(n_i, n_k)$ can be defined as

$$k_l(n_i, n_k) = \min_{\lambda_{n_i n_k} \in \Lambda_{n_i n_k}} c_l(\lambda_{n_i n_k}), \quad 0 \leq l \leq m.$$

We have $\vec{k}(n_i, n_k) = [k_0(n_i, n_k), k_1(n_i, n_k), \dots, k_m(n_i, n_k)]$, by convention.

In this work, we are mainly interested in paths that start from a given source node s. Hence, a path that starts from the node s and terminates at a node n is denoted by $\lambda_n = \langle n_0 \equiv s, n_1, n_2, \dots, n_i, \dots, n_k \equiv n \rangle$ for notational convenience. Any subpath of λ_n that starts from s and ends at n_i is denoted by $\lambda_{n_i/n} = \langle s, n_1, n_2, \dots, n_i \rangle$.

III. Problem Definition

This paper considers the problem of finding energy-constrained shortest paths as a resource constrained shortest path problem. Let us first consider a general resource constrained shortest path problem as a constrained search problem with m hard constraints. Given a locally finite digraph $\mathcal{G}$ and a vector of hard constraints $[\psi_1, \psi_2, \dots, \psi_m] \in \mathbb{R}^m$, a constrained search problem is to find a minimum c_0 cost path λ_t^* from s to t, *i.e.*

$$\lambda_t^* = \operatorname*{argmin}_{\lambda_t \in \Lambda_t} c_0(\lambda_t), \tag{1}$$

subjected to

$$c_l(\lambda_t^*) \leq \psi_l, \quad 1 \leq l \leq m. \tag{2}$$

The cost vector of λ_t^* can be denoted as $\vec{c}(\lambda_t^*)$.

Let a vector $\vec{\psi} = [\infty, \psi_1, \ldots, \psi_m]$. Then the constraints in (2) can be rewritten as $\vec{c}(\lambda_t^*) \preceq \vec{\psi}$ and violation of at least one of them can be denoted as $\vec{c}(\lambda_t^*) \npreceq \vec{\psi}$. Let a scalar $c_0^* = c_0(\lambda_t^*)$. Then we can define another vector $\vec{\psi}^* = [c_0^*, \psi_1, \ldots, \psi_m]$ such that $\vec{a} \in \mathbb{R}^{m+1}$, $\vec{a} \preceq \vec{\psi}^* \Rightarrow \vec{a} \preceq \vec{\psi}$. Now, we can redefine the above constrained search problem as finding a minimum c_0 cost path λ_t^* subjected to $\vec{c}(\lambda_t^*) \preceq \vec{\psi}^*$.

IV. CSA* Search Algorithm

Some requirements should be taken into account when designing a constrained search algorithm. Instead of scalar edge costs, the algorithm should be able to deal with vector edge costs. Moreover, the algorithm should be able to operate under more than one constraints, and when under multiple constraints, the algorithm should minimize the primary path cost without violating other constraints. One possible approach is to explore a digraph $\mathcal{G}$ uniformly using a uniform-cost search and try to minimize the primary path cost g_0. The node expansion process can be similar to that in Dijkstra's algorithm, while path cost vectors $\vec{g}$ can be obtained on-the-fly. Branches that have violated one or more constraints can be eliminated. Note that a path between s and n associated with the minimum g_0 value could have violated several constraints, thus it is regarded as infeasible. Meanwhile, there can be an alternative path that can fulfill all the constraints but with a higher g_0. To locate these feasible paths, the search will need to go back and explore options that were regarded as less desirable before. Such action can be time consuming and cause serious impacts to the performances of a path planning algorithm. A desirable constrained path planning algorithm should be able to look ahead for constraints violations and prune unpromising branches early. Nevertheless, without being guided by heuristics, the path planning algorithm cannot neglect any available options until such branch has encountered a violation and needed to be pruned.

*A. Operations of CSA**

In the proposed CSA* algorithm, path selection and expansion operations are used to avoid going back and re-expand optional branches which is a time-consuming process. Furthermore, intermediate search results are stored in a directed acyclic graph, namely a *search graph* [25] to ease the comparisons among paths with different properties. To reach a point t on the map, the expected cost for the corresponding complete path via expanding an intermediate path λ_n is denoted as

$$\vec{f}(\lambda_n) = [f_0(\lambda_n), f_1(\lambda_n), \ldots, f_m(\lambda_n)] = \vec{g}(\lambda_n) + \vec{h}(n). \quad (3)$$

The corresponding cost vector of the intermediate path λ_n is expressed as $\vec{g}(\lambda_n) = [g_0(\lambda_n), g_1(\lambda_n), \ldots, g_m(\lambda_n)]$, where

$$g_l(\lambda_n) = c_l(\lambda_n), \quad 0 \le l \le m.$$

Here, g_0 is the *primary cost* and g_l $(1 \le l \le m)$ are representing the m *constrained costs*. In (3), $\vec{h}(n) = [h_0(n), h_1(n), \ldots, h_m(n)]$ is a vector of the heuristic costs, where h_0 and h_l $(1 \le l \le m)$ respectively denote the *primary* and the *constrained heuristic cost estimations* from n to t.

Algorithm 1: CSA* search algorithm

```
Step 1: If $\vec{h}(s) \npreceq \vec{\psi}$, then exit with failure.
Step 2: Record $\xi(\lambda_s) = \{s, \vec{0}, \vec{h}(s), \mathrm{NULL}\}$ on OPEN.
Step 3: If OPEN is empty, then exit with failure.
Step 4: Remove $\xi(\lambda_{n_i}) = \{n_i, \vec{g}(\lambda_{n_i}), \vec{f}(\lambda_{n_i}), p(\lambda_{n_i})\}$ from
        OPEN whose $f_0$ cost is minimum and record it on
        CLOSED. If there exsit more than one such entries,
        select an entry among them such that its $\vec{f}$ dominates
        or equals others. Select a path arbitarary if they
        are nondominated to each other, but favor any path
        terminating at $t$.
Step 5: If $n_i$ is the target node, i.e. $n_i = t$, then exit with the
        path obtained by tracing back pointers from $p(\lambda_t)$ to
        NULL.
Step 6: Otherwise, for each successor $n_{i+1}$ of $n_i$ that do not
        produces cycles in the search graph:
          a) Calculate $\vec{g}(\lambda_{n_{i+1}})$ and $\vec{f}(\lambda_{n_{i+1}})$.
          b) If $\vec{f}(\lambda_{n_{i+1}}) \npreceq \vec{\psi}$, then prune $\lambda_{n_{i+1}}$ and go to Step
             6a.
          c) If there exists a path $\lambda'_{n_{i+1}}$ on OPEN or CLOSED
             such that $\vec{f}(\lambda'_{n_{i+1}}) \preceq \vec{f}(\lambda_{n_{i+1}})$, then prune $\lambda_{n_{i+1}}$
             and go to Step 6a.
          d) If $\lambda_{n_{i+1}}$ dominates any paths from $s$ to $n_{i+1}$
             which are already on OPEN, prune all such paths
             and remove corresponding entries from OPEN.
          e) Set $p(\lambda_{n_{i+1}}) = \lambda_{n_i}$ and record $\xi(\lambda_{n_{i+1}}) =$
             $\{n_{i+1}, \vec{g}(\lambda_{n_{i+1}}), \vec{f}(\lambda_{n_{i+1}}), p(\lambda_{n_{i+1}})\}$ on OPEN.
Step 7: Go to Step 3.
```

The proposed algorithm makes better expanding decisions by avoiding branches that is certainly not a part of the optimal path and not giving up any subpaths that have potentials to be included in the optimal path. While the ideas follow the basic principles of A*, it is a nontrivial task to make the above decisions as the process involves multiple cost estimations and continuously checking for constraint violations.

Like ordinary A* and its successors, the proposed algorithm records path information using CLOSED and OPEN lists. However, their purposes are redefined. For paths that their costs have been calculated but are not yet expanded, they are stored in OPEN. In contrast, paths that have been expanded and will not be expanded again are stored in CLOSED. Therefore, for all the paths stored in OPEN, their subpaths that originated from s should have been expanded and stored in CLOSED. To store a path $\lambda_{n_i} = \langle s, n_1, n_2, \ldots, n_{i-1}, n_i \rangle$ in these lists, its information is retained as a four element entry: the ending node n_i, two cost vectors $\vec{g}(\lambda_{n_i})$ and $\vec{f}(\lambda_{n_i})$, and a pointer to its preceding path $p(\lambda_{n_i})$. Such an entry is denoted as

$$\xi(\lambda_{n_i}) = \{n_i, \vec{g}(\lambda_{n_i}), \vec{f}(\lambda_{n_i}), p(\lambda_{n_i})\},$$

where $p(\lambda_{n_i}) = \lambda_{n_{i-1}}$.

Algorithm 1 provides a summary of the proposed algorithm. CSA* initializes with the calculation of the heuristic cost vector $\vec{h}(s)$. Since all paths originate from s, $\vec{g}(\lambda_s) = 0$ and therefore, according to (3), $\vec{f}(\lambda_s) = \vec{h}(s)$. At this point, any path starting from s is expected to violate at least one of the constraints if $\vec{h}(s) \npreceq \vec{\psi}$. On the other hand, if no violations are expected, *i.e.* $\vec{h}(s) \preceq \vec{\psi}$ in Step 2, $\xi(\lambda_s)$ is then stored in OPEN. Note that as λ_s has no preceding path segment, its pointer $p(\lambda_s) = \text{NULL}$. This completes the initialization of the proposed CSA*. Steps 3-7 of the proposed algorithm are then executed iteratively until an optimal solution is found (Step 5). If a feasible solution does not exist, it terminates without a solution (Step 3).

The path selection mechanism in CSA* is very different from that of NAMOA* as it selects an entry $\xi(\lambda_{n_i})$ on OPEN list that has the minimum f_0 cost among all other entries on OPEN (Step 4). This path selection procedure is comparable to the node selection procedure in A*.

Path expansion procedure of CSA* is summarized in Step 6. For each successor n_{i+1} of n_i, CSA* calculates costs of extending path λ_{n_i} to n_{i+1} as

$$\vec{g}(\lambda_{n_{i+1}}) = \vec{g}(\lambda_{n_i}) + \vec{c}(n, n_{i+1}),$$
$$\vec{f}(\lambda_{n_{i+1}}) = \vec{g}(\lambda_{n_{i+1}}) + \vec{h}(n_{i+1}),$$

in Step 6a. If $\vec{f}(\lambda_{n_{i+1}}) \npreceq \vec{\psi}$, *i.e.* any constraints are expected to be violated by the extended path $\lambda_{n_{i+1}}$, it is pruned and returns to n_i in Step 6b to evaluate its next successor. The early termination of CSA* given in Step 1 is a special case of path pruning given in Step 6b. The computational efficiency of CSA* can be considerably improved without compromising its admissibility by path pruning with accurate heuristic estimations. Like other heuristic search algorithms, the performances of the proposed CSA* depend on the accuracy of its heuristic cost estimations. The path $\lambda_{n_{i+1}}$ can be considered as a subpath of a potential optimal path leading to t if $\vec{f}(\lambda_{n_{i+1}}) \preceq \vec{\psi}$. CSA* sets $p(\lambda_{n_{i+1}}) = \lambda_{n_i}$ and stores $\xi(\lambda_{n_{i+1}})$ in OPEN to be expanded later (Step 6e) if $\lambda_{n_{i+1}}$ is the only path discovered to n_{i+1} so far. Nevertheless, storing $\xi(\lambda_{n_{i+1}})$ in OPEN might be skipped if CSA* has already discovered another path leading to n_{i+1}. CSA* checks whether $\lambda_{n_{i+1}}$ is dominated by $\lambda'_{n_{i+1}}$ if it finds a path $\lambda'_{n_{i+1}}$ on either OPEN or CLOSED.

Definition 1: A path λ_n is said to be *dominated* if

$$\exists \lambda'_n \in \Lambda_n \quad | \quad \vec{f}(\lambda'_n) \prec \vec{f}(\lambda_n). \tag{4}$$

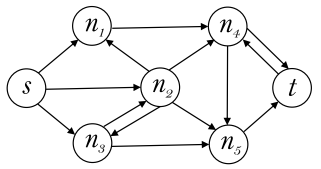
On the contrary, λ_n is said to be *nondominated* if

$$\nexists \lambda'_n \in \Lambda_n \quad | \quad \vec{f}(\lambda'_n) \prec \vec{f}(\lambda_n). \tag{5}$$

CSA* prunes $\lambda_{n_{i+1}}$ and returns to n_i to evaluate its next successor if it finds any path $\lambda'_{n_{i+1}}$ that dominates or equals $\lambda_{n_{i+1}}$, *i.e.* $\vec{f}(\lambda'_{n_{i+1}}) \preceq \vec{f}(\lambda_{n_{i+1}})$ (Step 6c). On the other hand, CSA* prunes all such dominated paths if CSA* finds any path that is already on OPEN and dominated by $\lambda_{n_{i+1}}$ (Step 6d). Moreover, CSA* sets $p(\lambda_{n_{i+1}}) = \lambda_{n_i}$ and stores $\xi(\lambda_{n_{i+1}})$ in OPEN (Step 6e).

B. An Illustrative Case Study

An illustrative case study is provided in this subsection to ease the elaboration of the proposed CSA* algorithm. It is based on an arbitrary digraph $\mathcal{G}$ shown in Fig. 1, which $\mathcal{G}$ comprises 7 nodes and 14 edges connecting them. The corresponding edge cost vectors ($\vec{c}$) and heuristic cost vectors ($\vec{h}$) are shown in the same figure. The cost associated with a path λ_t between a source node s and a target node t is having a cost vector $\vec{g}(\lambda_t) = [g_0(\lambda_t), g_1(\lambda_t), g_2(\lambda_t)]$. If the constrained optimization problem is relaxed into its unconstrained version (*i.e.* minimizing g_0 solely), it renders $\lambda_t = \langle s, n_1, n_4, t\rangle$ to be a feasible solution with its $\vec{g}(\lambda_t) = [7, 9, 9]$. If constraints $g_1 \leq 8$ and $g_2 \leq 9$ are imposed, *i.e.* $\psi = [\infty, 8, 9]$, the previous solution is not feasible as the first constraint is violated.



n	$\vec{h}(n)$
s	$[6, 5, 7]$
n_1	$[5, 4, 7]$
n_2	$[3, 2, 6]$
n_3	$[4, 2, 7]$
n_4	$[2, 2, 4]$
n_5	$[1, 0, 2]$
t	$[0, 0, 0]$

(n_i, n_j)	$\vec{c}(n_i, n_j)$
(s, n_1)	$[1, 2, 1]$
(s, n_2)	$[3, 3, 2]$
(s, n_3)	$[2, 3, 1]$
(n_1, n_4)	$[3, 2, 3]$
(n_2, n_1)	$[4, 2, 3]$
(n_2, n_3)	$[3, 2, 4]$
(n_2, n_4)	$[4, 0, 3]$
(n_2, n_5)	$[3, 5, 3]$
(n_3, n_2)	$[0, 2, 3]$
(n_3, n_5)	$[4, 4, 4]$
(n_4, n_5)	$[2, 1, 1]$
(n_4, t)	$[3, 5, 5]$
(n_5, t)	$[2, 2, 4]$
(t, n_4)	$[3, 2, 1]$

Fig. 1. A diagraph (at top left) with its edge cost vectors (at right) and heuristic cost vectors (at bottom left).

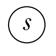
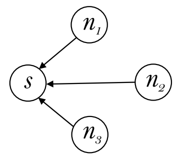
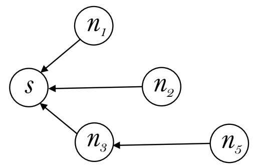
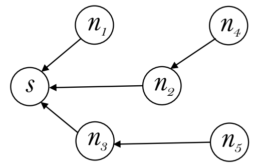
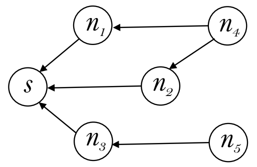
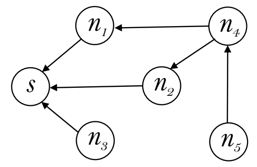
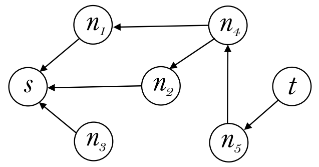
The following procedural execution of the proposed CSA* illustrates how an optimal solution is obtained under some given constraints. The step-wise evolution of a search graph and the changes in OPEN and CLOSED lists are given in TABLE I. In Step 1, the proposed algorithm begins with the evaluation of $\vec{h}(s)$. As $\vec{h}(s) \preceq \psi$, in Step 2, $\xi(\lambda_s) = \{s, [0, 0, 0], [6, 5, 7], \text{NULL}\}$ is stored as the first entry in OPEN where $\lambda_s = \langle s\rangle$. Starting from Step 3, the proposed CSA* proceeds to find the optimal solution iteratively.

In the first iteration, CSA* removes $\xi(\lambda_s)$ from OPEN and stored it in CLOSED (Step 4). As $s \neq t$ in this case study, successors of s, *i.e.* n_1,n_2, and n_3, are generated in Step 6. Here, we denote paths $\lambda_{n_1} = \langle s, n_1\rangle$, $\lambda''_{n_3} = \langle s, n_3\rangle$, and $\lambda'_{n_2} = \langle s, n_2\rangle$ accordingly. Their corresponding cost vectors are therefore obtained in Step 6a as follows.

$$\vec{f}(\lambda_{n_{i+1}}) = (\vec{g}(\lambda_{n_i}) + \vec{c}(n_i, n_{i+1})) + \vec{h}(n_{i+1}),$$
$$\vec{f}(\lambda_{n_1}) = ([0, 0, 0] + [1, 2, 1]) + [5, 4, 7] = [6, 6, 8],$$
$$\vec{f}(\lambda''_{n_3}) = ([0, 0, 0] + [2, 3, 1]) + [4, 2, 7] = [6, 5, 8],$$
$$\vec{f}(\lambda'_{n_2}) = ([0, 0, 0] + [3, 3, 2]) + [3, 2, 6] = [6, 5, 8].$$

At the moment, OPEN is empty and none of the cost vectors violates any of the constraints, thus all three paths do not need to undergo any pruning. Consequently, in Step 6e, OPEN is appended with predecessors $\xi(\lambda_{n_1})$, $\xi(\lambda'_{n_2})$, and $\xi(\lambda''_{n_3})$. As

TABLE I
ILLUSTRATIONS OF THE SEARCH GRAPH AND OPEN AND CLOSED LISTS AFTER EACH ITERATION.

Itera.	Search graph	OPEN	CLOSED
0		$\xi(\lambda_s)$	
1		$\xi(\lambda''_{n_3})$ $\xi(\lambda'_{n_2})$ $\xi(\lambda_{n_1})$	$\xi(\lambda_s)$
2		$\xi(\lambda'_{n_2})$ $\xi(\lambda_{n_1})$ $\xi(\lambda''_{n_5})$	$\xi(\lambda_s)$ $\xi(\lambda''_{n_3})$
3		$\xi(\lambda_{n_1})$ $\xi(\lambda''_{n_5})$ $\xi(\lambda'_{n_4})$	$\xi(\lambda_s)$ $\xi(\lambda''_{n_3})$ $\xi(\lambda'_{n_2})$
4		$\xi(\lambda_{n_4})$ $\xi(\lambda''_{n_5})$ $\xi(\lambda'_{n_4})$	$\xi(\lambda_s)$ $\xi(\lambda''_{n_3})$ $\xi(\lambda'_{n_2})$ $\xi(\lambda_{n_1})$
5		$\xi(\lambda_{n_5})$ $\xi(\lambda'_{n_4})$	$\xi(\lambda_s)$ $\xi(\lambda''_{n_3})$ $\xi(\lambda'_{n_2})$ $\xi(\lambda_{n_1})$ $\xi(\lambda_{n_4})$
6		$\xi(\lambda_t)$ $\xi(\lambda'_{n_4})$	$\xi(\lambda_s)$ $\xi(\lambda'_{n_3})$ $\xi(\lambda_{n_1})$ $\xi(\lambda_{n_4})$ $\xi(\lambda_{n_5})$

shown in TABLE I, the search graph is expended with 3 extra nodes by the end of the first iteration. The predecessor of a node is indicated by the arrowhead.

At the beginning of the next iteration, all entries on OPEN are having their f_0 costs equal 6. As a result, in Step 4, the proposed algorithm will prefer entries with non-dominated cost vector $\vec{f}$ over their counterparts. Certainly, both $\xi(\lambda'_{n_2})$ and $\xi(\lambda''_{n_3})$ are desirable candidates due to the fact that $\vec{f}(\lambda'_{n_2}) = \vec{f}(\lambda''_{n_3}) \prec \vec{f}(\lambda_{n_1})$. Arbitrarily, suppose $\xi(\lambda''_{n_3})$ is chosen to be expended over $\xi(\lambda'_{n_2})$. Then $\xi(\lambda''_{n_3})$ is removed from OPEN and stored in CLOSED in Step 4. All successors of n_3 are then generated while their cost vectors are obtained in Step 6a as follows.

$$\vec{f}(\lambda''_{n_5}) = ([2,3,1] + [4,4,4]) + [1,0,2] = [7,7,7],$$
$$\vec{f}(\lambda''_{n_2}) = ([2,3,1] + [0,2,3]) + [3,2,6] = [5,7,10].$$

In Step 6b, $\lambda''_{n_2} = \langle s, n_3, n_2\rangle$ is pruned because $\vec{f}(\lambda''_{n_2}) \npreceq \psi$. So far, $\lambda''_{n_5} = \langle s, n_3, n_5\rangle$ is the only path that can satisfy all constraints and reach n_5. Therefore, $\xi(\lambda''_{n_5})$ is stored in OPEN with $p(\lambda''_{n_5}) = \lambda''_{n_3}$.

OPEN holds three entries by the beginning of the third iteration. They are $\xi(\lambda'_{n_2})$, $\xi(\lambda_{n_1})$, and $\xi(\lambda''_{n_5})$, where $f_0(\lambda'_{n_2}) = f_0(\lambda_{n_1}) < f_0(\lambda''_{n_5})$ and $\vec{f}(\lambda'_{n_2}) \prec \vec{f}(\lambda_{n_1})$. Therefore, the entry $\xi(\lambda'_{n_2})$ is removed from OPEN and stored in CLOSED (Step 4). The cost vectors of n_2's successors are calculated as

$$\vec{f}(\lambda'_{n_3}) = [6,5,6] + [4,2,7] = [10,7,13],$$
$$\vec{f}(\lambda'_{n_4}) = [7,3,5] + [2,2,4] = [9,5,9],$$
$$\vec{f}(\lambda'_{n_1}) = [7,5,5] + [5,4,7] = [12,9,12],$$
$$\vec{f}(\lambda'_{n_5}) = [6,8,5] + [1,0,2] = [7,8,7].$$

Since both $\vec{f}(\lambda'_{n_1})$ and $\vec{f}(\lambda'_{n_3})$ violate the constraints, their corresponding paths $\lambda'_{n_1} = \langle s, n_2, n_1\rangle$ and $\lambda'_{n_3} = \langle s, n_2, n_3\rangle$ are pruned in Step 6b. Since $\vec{f}(\lambda''_{n_5}) \prec \vec{f}(\lambda'_{n_5})$, CSA* prunes $\lambda'_{n_5} = \langle s, n_2, n_5\rangle$ in Step 6c. Lastly, $\xi(\lambda'_{n_4})$ is stored in OPEN in Step 6e as $\lambda'_{n_4} = \langle s, n_2, n_4\rangle$ is the only remaining path of which $p(\lambda'_{n_4}) = \lambda'_{n_2}$.

In the following iteration, node n_1 is chosen to be expended in Step 4 due to its smallest f_0 among all entries on OPEN. Afterward, $\xi(\lambda_{n_1})$ is removed from OPEN and stored in CLOSED. The cost vector of n_2's only successor is obtained as follow.

$$\vec{f}(\lambda_{n_4}) = [4,4,4] + [2,2,4] = [6,6,8].$$

Note that as λ_{n_4} and λ'_{n_4} are non-dominated by each other, thus $\xi(\lambda_{n_4})$ of which $p(\lambda_{n_4}) = \lambda_{n_1}$, is stored in OPEN. As shown in Fig. 1, at the moment, n_4 can be reached by either $\lambda'_{n_4} = \langle s, n_2, n_4\rangle$ or $\lambda_{n_4} = \langle s, n_1, n_4\rangle$. Their cost vectors are $[9,5,9]$ and $[6,6,8]$, respectively.

In the fifth iteration, due to the criteria in Step 4, $\xi(\lambda_{n_4})$ is removed from OPEN and stored in CLOSED. Cost vectors of n_4's successors are obtained as follows.

$$\vec{f}(\lambda_{n_5}) = [6,5,5] + [1,0,2] = [7,5,7],$$
$$\vec{f}(\lambda'''_{t}) = [7,9,9] + [0,0,0] = [7,9,9].$$

Path $\lambda'''_t = \langle s, n_1, n_4, t\rangle$ is pruned in Step 6b due a constraint violation. Back in the second iteration, n_5 can already be accessed via λ''_{n_5} with $\vec{f} = [7,7,7]$. In the current iteration, n_5 is also reachable via $\lambda_{n_5} = \langle s, n_1, n_4, n_5\rangle$ with $\vec{f} = [7,5,7]$.

As $\lambda_{n_5} \prec \lambda''_{n_5}$, λ''_{n_5} is pruned in Step 6d. Subsequently, in Step 6e, $\xi(\lambda''_{n_5})$ is discarded from OPEN and $\xi(\lambda_{n_5})$ is stored in OPEN with $p(\lambda_{n_5}) = \lambda_{n_4}$.

The sixth iteration begins with having $\xi(\lambda'_{n_4})$ and $\xi(\lambda_{n_5})$ on OPEN, with their cost vectors equal $[9,5,9]$ and $[7,5,7]$, respectively. By comparing their f_0 values, $\xi(\lambda_{n_5})$ is removed from OPEN and stored in CLOSED in Step 4. The cost vector of the path $\lambda_t = \langle s, n_1, n_4, n_5, t\rangle$ is obtained as follow.

$$\vec{f}(\lambda_t) = [8,7,9] + [0,0,0] = [8,7,9].$$

Under any criteria in Step 6, λ_t cannot be pruned. Therefore, $\xi(\lambda_t)$ is stored in OPEN with $p(\lambda_t) = \lambda_{n_5}$.

At the moment, $\xi(\lambda'_{n_4})$ and $\xi(\lambda_t)$ are the only entries remaining in OPEN. In the seventh iteration, $\xi(\lambda_t)$ is removed from OPEN and stored in CLOSED because $f_0(\lambda_t) < f_0(\lambda'_{n_4})$. Note that $\xi(\lambda_t)$ satisfies the criterion in Step 5, thus the search process terminates by CAS* reassembling the optimal path by tracing the pointers from $p(\lambda_t)$ to NULL. The optimal path that satisfies all given constraints is obtained as $\lambda^*_t = \langle s, n_1, n_4, n_5, t\rangle$ of which $\vec{g}(\lambda^*_t) = \vec{f}(\lambda^*_t) = [8,7,9]$.

V. Energy-Constrained Shortest Path Planning

The proposed CSA* is adopted for energy-constrained shortest path planning on uneven terrains and, in this section, its performances are compared with state-of-the-art uneven terrain path planning algorithms.

A. Test Setup

Two tests, namely Test I and Test II, were conducted using elevation maps of Matheny ridge and Anderson canyon in USA. In order to facilitate the path planning process, a selected elevation map is first transformed into a finite digraph as proposed in [18]. Distance- and energy-costs associated with the traversal between two nodes n and n' are given by

$$c_d(n,n') = \begin{cases} \infty, & \text{if } \phi(n,n') > \phi_m, \\ s(n,n'), & \text{otherwise,} \end{cases} \tag{6}$$

and

$$c_e(n,n') = \begin{cases} \infty, & \text{if } \phi(n,n') > \phi_m, \\ mgs(n,n')(\mu\cos\phi(n,n') + \sin\phi(n,n')), & \\ & \text{if } \phi_m \geq \phi(n,n') > \phi_b, \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

respectively [24]. All tests were conducted using a simulated model platform whose mass $m = 300$ kg. It can operate at a maximum motion power of 1280 W. In (6) and (7), $s(n,n')$ is the Euclidean distance between n and n', $\phi(n,n')$ is the elevation angle between n and n', ϕ_m is the critical impermissible angle, and ϕ_b is the breaking angle. Interested readers may refer to the computations of ϕ_m and ϕ_b given in [24]. The rest of the parameter values are adopted from [24], including the friction coefficient $\mu = 0.01$ and the gravitational field strength $g = 9.81$ ms^{-2}.

Using (6) and (7), the cost vector associated with nn' is defined as

$$\vec{c}(n,n') = [c_d(n,n'), c_e(n,n')].$$

The heuristic cost vector for cost estimation from n to the target is defined as

$$\vec{h}(n) = [h_d(n), h_e(n)],$$

where

$$h_d(n) = s(n,t),$$

and

$$h_e(n) = \begin{cases} \frac{mg\Delta(n,t)}{\sin\phi_m}(\mu\cos\phi_m + \sin\phi_m), & \\ & \text{if } \phi(n,t) > \phi_m, \\ mgs(n,t)(\mu\cos\phi\ (n,t) + \sin\phi(n,t)), & \\ & \text{if } \phi_m \geq \phi(n,t) > \phi_b, \\ 0, & \text{otherwise.} \end{cases}$$

The consistency of $h_d(n)$ and $h_e(n)$ has been proven in [24]. Finally, a vector of hard constraints is defined as

$$\vec{\psi} = [\infty, \psi_e],$$

where ψ_e is the maximum available energy of a mobile platform. The values of ψ_e is given in TABLES III and V.

B. Test Results

In Test I, the path planning task was to plan a feasible shortest path from $(210, 310, 310.6)$ m to $(720, 880, 373.5)$ m on a section of Matheny ridge. The mobile platform is set to travel at 0.7 m/s with an additional payload of 75 kg. According to the results given in TABLE II, the length of the shortest path planned using Dijkstra's algorithm is 837.715 m and the mobile platform consumes 538.053 kJ to traverse it. If the available energy is less than that, say 450 kJ, such a path cannot be traversed. On the other hand, the energy-optimal path obtained using Z* requires only 274.629 kJ. However, it is nearly 12% longer than the shortest path. NAMOA* based multiobjective path planner proposed in [24] is capable of finding all nondominated paths between two given points. All nondominated paths generated in this test and their cost values are illustrated in Figs. 2 and 3, respectively. As expected, two of these nondominated paths coincide with the shortest and energy-optimal paths. Now it is possible to select the shortest path that satisfy a given energy constraint out of these nondominated paths. However, finding all nondominated paths is computationally expensive, thus, not feasible in time-critical applications. Here, NAMOA* has expanded 14712 subpaths for finding all nondominated paths. While given a constraint, the proposed CSA* algorithm can find the shortest path by expanding a minimum number of subpaths. As an example given in TABLE III, when the maximum available energy is 450kJ, CSA* has found a constraint satisfying shortest path by expanding only 2833 subpaths. Computational efficiency of search algorithms are measured in terms of number of expanded subpaths. Thus, finding a constraint satisfying path using CSA* in this particular case is over 5 times more efficient than using NAMOA*. Notably, when the energy constraint is relaxed, *i.e.* $\psi_e = \infty$, CSA* finds the globally shortest path. Under tight energy constraints such as

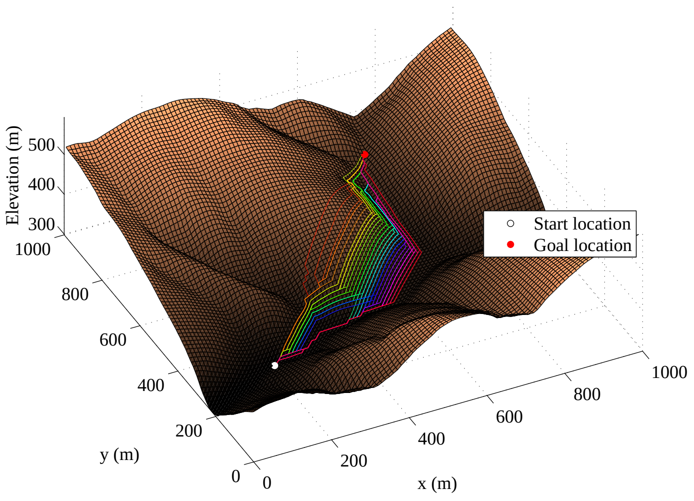


Fig. 2. All nondominated paths obtained using NAMOA* in Test I.

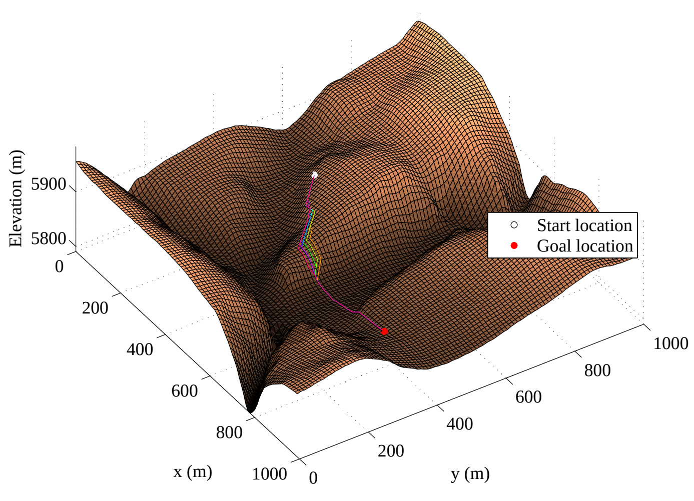


Fig. 4. All nondominated paths obtained using NAMOA* in Test II.

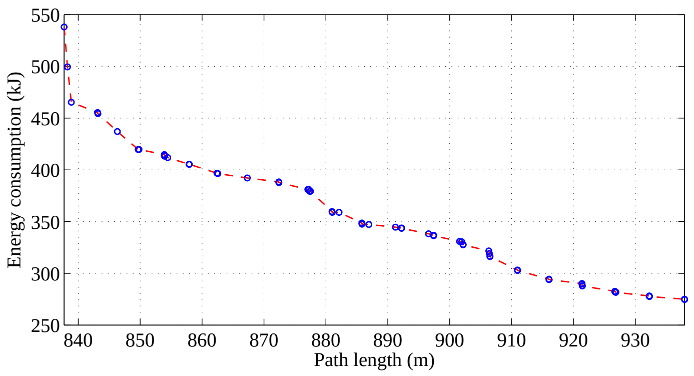


Fig. 3. A Pareto frontier that represents path lengths and energy consumptions of the nondominated paths given in Fig. 2.

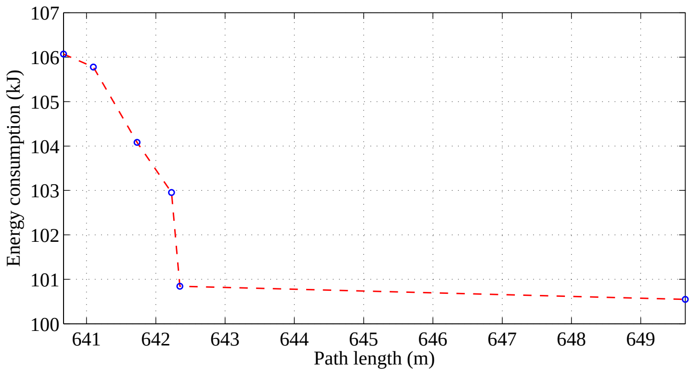


Fig. 5. A Pareto frontier that represents path lengths and energy consumptions of the nondominated paths given in Fig. 4.

TABLE II
RESULTS OF TEST I USING DIJKSTRA'S AND Z*.

Algorithm	Dijkstra's	Z*
Path length (m)	837.715	937.927
Energy consumption (kJ)	538.053	274.629

TABLE IV
RESULTS OF TEST I USING DIJKSTRA'S AND Z*.

Algorithm	Dijkstra's	Z*
Path length (m)	640.668	649.643
Energy consumption (kJ)	106.070	100.550

TABLE III
RESULTS OF TEST I USING CSA*.

Maximum energy (kJ)	274.630	350.000	450.000	Unlimited
Path length (m)	937.927	885.820	846.326	837.715
Energy consumption (kJ)	274.629	348.661	437.040	538.053
# of sub-paths explored	2446	4745	2833	2507

TABLE V
RESULTS OF TEST II USING CSA*.

Maximum energy (kJ)	100.600	103.000	104.500	Unlimited
Path length (m)	649.643	642.227	641.729	640.668
Energy consumption (kJ)	100.550	102.955	104.081	106.070
# of sub-paths explored	1523	1257	1304	1984

$\psi_e = 274.630$ kJ, CSA* finds the same energy-optimal path as Z* does.

In Test II, the objective was to plan a feasible shortest path from $(260, 540, 5892)$ m to $(820, 380, 5858)$ m on a section of Anderson canyon. The mobile platform is set to travel at 0.6 m/s without any payloads. According to the results given in TABLE IV, the length of the shortest path is 640.668 m and the mobile platform consumes 106.069 kJ to traverse it. On the other hand, the energy-optimal path obtained using Z* consumes only 100.549 kJ, but it is longer than the shortest path. All nondominated paths generated in this test and their cost values are illustrated in Figs. 4 and 5, respectively. In this test, NAMOA* has expanded 57252 subpaths to find all nondominated paths. According to the results given in TABLE V, CSA* finds constraints satisfying paths more efficiently compared to NAMOA*. For example, CSA* is over 45 times more efficient than NAMOA* for finding a shortest path with a maximum of 103 kJ of energy.

VI. Discussion

Properties of CSA*, including its strengths, limitations, and possible improvements, are discussed in details in this section. Rigorous theoretical analyses on both the admissibility and efficiency of CSA* are provided in appendices. When CSA* is guided by admissible heuristics, it is proven to find an optimal solution under constraints if such a solution exists. CSA* guided by consistent heuristics, is proven to be optimal over a class of equally informed admissible constrained search algorithms with respect to both the number of paths and the set of paths expanded. The search efficiency of CSA* can be improved by using more informed heuristics. As shown in appendices, heuristics govern both the optimality and admissibility of CSA*.

Rather than considering CSA* as a single algorithm, it can be considered as a family of algorithms. Its search behavior can vary from one application to another based on the nature of heuristics and the number of constraints. Heuristics can be somehow interpreted as the knowledge of the problem domain. In the absence of it, *i.e.* $\vec{h} = \vec{0}$, CSA* becomes a uniform-cost search. Albeit $\vec{h} = \vec{0}$ being the least informed heuristic vector, it is consistent. Therefore, it preserves both the optimality and admissibility of CSA*. In many practical applications, however, more informed heuristics can be found easily. If such estimations cannot be found for some constrained heuristics, those particular heuristics in a heuristic vector can be set to zero while the rest of the vector remain non-zero. Such a heuristic vector can still guarantee both the optimality and admissibility of CSA* since the complete heuristic vector is consistent.

If any component of a heuristic vector overestimates the cost of an optimal path, *i.e.* $\exists n \in \mathcal{N} \mid \vec{h}(n) \npreceq \vec{h}^*(n)$, then CSA* guided by such heuristics is no longer admissible. Nevertheless, CSA* guided by such non-admissible heuristics can sometimes find a solution faster than one guided by admissible heuristics, by expanding a smaller number of paths. However, because of path pruning in Steps 1 and 6b in Algorithm 1 due to mispredicted constraint violations, CSA* guided by such non-admissible heuristics might fail to find a solution even if a solution exists. Therefore, the utilization of non-admissible constrained heuristics to accelerate CSA* must be carried out with cautions. Nevertheless, non-admissible primary heuristics can possibly accelerate CSA* without generating such false alarms but may return sub-optimal solutions.

VII. Conclusion

This paper proposed CSA* for solving constrained search problems. CSA* is capable of accommodating multiple constraints. CSA* can be implemented easily and analyzed rigorously as ordinary A*. If there exists a solution for a given constrained search problem, CSA* guided by admissible heuristics guarantees to find an optimal solution that satisfy the given constraints. If the heuristics are consistent, CSA* is proven to be optimal with respect to the set of paths expanded over a class of equally informed admissible constrained search algorithms. More informed heuristics can improve the search efficiency of CSA*. Test results provided in this paper suggest that CSA* is suitable for shortest path planning for energy-constrained mobile platforms on uneven terrains.

In many real-world applications, mobile agents have to re-plan their paths due to unforeseen disruptions. Path replanning from scratch can be computationally very expensive, thus may interrupt the seamless operation of mobile agents if it has to be executed regularly. Therefore, future work should investigate on admissible and efficient algorithms for path replanning under multiple constraints.

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc., Reading, MA, 1984, ch. Formal properties of heuristic methods, pp. 73–85.

[4] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car – Part I: Distributed system architecture and development process," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 7131–7140, 2014.

[5] L. Gomes, "When will Google's self-driving car really be ready? It depends on where you live and what you mean by "ready"," *IEEE Spectrum*, vol. 53, no. 5, pp. 13–14, 2016.

[6] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017.

[7] H. Mahboubi, A. G. Aghdam, and K. Sayrafian-Pour, "Toward autonomous mobile sensor networks technology," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 576–586, 2016.

[8] N. Ganganath, C.-T. Cheng, and C. K. Tse, "Distributed Anti-flocking Algorithms for Dynamic Coverage of Mobile Sensor Networks," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1795–1805, 2016.

[9] N. Ganganath, W. Yuan, T. Fernando, H. H. C. Iu, and C.-T. Cheng, "Energy-efficient anti-flocking control for mobile sensor networks on uneven terrains," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. PP, no. 99, 2018.

[10] G. V. Levin, "The curiousness of Curiosity," *Astrobiology*, vol. 15, no. 2, pp. 101–103, 2015.

[11] M. Sutoh, M. Otsuki, S. Wakabayashi, T. Hoshino, and T. Hashimoto, "The right path: comprehensive path planning for lunar exploration rovers," *IEEE Robotics & Automation Magazine*, vol. 22, no. 1, pp. 22–33, 2015.

[12] N. Ganganath, C.-T. Cheng, and C. K. Tse, "Rapid replanning of energy-efficient paths for navigation on uneven terrains," in *International Confonference on Industrial Informatics (INDIN)*. IEEE, 2015, pp. 408–413.

[13] W. Yuan, N. Ganganath, C.-T. Cheng, Q. Guo, and F. C. M. Lau, "A consistent heuristic for efficient path planning on mobility maps," in *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2017, pp. 1–5.

[14] N. G. Marasinghe Arachchige, "Heuristic search algorithms with applications to path planning on uneven terrains," Ph.D. dissertation, The Hong Kong Polytechnic University, 2016.

[15] M. Lanthier, A. Maheshwari, and J.-R. Sack, "Shortest anisotropic paths on terrains," in *Automata, Languages and Programming*. Springer, 1999, pp. 524–533.

[16] N. Ganganath, C.-T. Cheng, and C. K. Tse, "Rapidly replanning A*," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, 2016, pp. 386–389.

[17] K.-L. Wu, T.-J. Ho, S. A. Huang, K.-H. Lin, Y.-C. Lin, and J.-S. Liu, "Path planning and replanning for mobile robot navigation on 3Dterrain: An approach based on geodesic," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[18] N. Ganganath, C.-T. Cheng, and C. K. Tse, "A constraint-aware heuristic path planner for finding energy-efficient paths on uneven terrains," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 601–611, 2015.

[19] N. C. Rowe and R. S. Ross, "Optimal grid-free path planning across arbitrarily contoured terrain with anisotropic friction and gravity effects," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 5, pp. 540–553, 1990.

[20] N. C. Rowe and Y. Kanayama, "Near-minimum-energy paths on a vertical-axis cone with anisotropic friction and gravity effects," *The International Journal of Robotics Research*, vol. 13, no. 5, pp. 408–433, 1994.

[21] Q. Yuan, Q. Lu, and Z. Xi, "Optimal path selection for mobile robots based on energy consumption assessment of different terrain surface," in *36th Chinese Control Conf. (CCC)*. IEEE, 2017, pp. 6755–6760.

[22] N. Ganganath, C.-T. Cheng, and C. K. Tse, "An improved Dynamic Z* algorithm for rapid replanning of energy-efficient paths," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, 2015, pp. 395–398.

[23] Z. Sun and J. H. Reif, "On finding energy-minimizing paths on terrains," *IEEE Transactions on Robotics*, vol. 21, no. 1, pp. 102–114, 2005.

[24] N. Ganganath, C.-T. Cheng, and C. K. Tse, "Multiobjective path planning on uneven terrains based on NAMOA*," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 1846–1849.

[25] L. Mandow and J. L. P. De La Cruz, "Multiobjective A* search with consistent heuristics," *Journal of the ACM (JACM)*, vol. 57, no. 5, p. 27, 2010.

Appendix A
Admissibility of CSA*

In this section, we prove the admissibility of CSA* under some common assumptions. Like all other heuristically guided search algorithms, admissibility of the proposed CSA* algorithm depends on the properties of its heuristics. Here we introduce a special heuristic cost vector $\vec{h}^*(n) = [h_0^*(n), h_1^*(n), \ldots, h_m^*(n)]$, where a heuristic function $h_l^*(n) = k_l(n,t)$ for $0 \leq l \leq m$, *i.e.*

$$\vec{h}^*(n) = \vec{k}(n,t).$$

This implies that $h_0^*(n) \leq c_0^*$ for any node n on λ_t^*.

Definition 2: A heuristic cost vector $\vec{h}(n)$ is said to be *admissible* if

$$\vec{h}(n) \preceq \vec{h}^*(n), \quad \forall n \in \mathcal{N}. \tag{8}$$

Here, we assume that CSA* is always guided by admissible heuristic cost vectors. Similar assumptions are commonly used to prove the admissibility of other heuristic search algorithms [2], [3], [25].

CSA* is forced for an early termination in Step 1 if $\vec{h}(s) \npreceq \vec{\psi}$. Therefore, it is essential to guarantee that such an exit occurs only when there is no solution available for a given problem.

Theorem 1: If there exists a solution, then $\vec{h}(n) \preceq \vec{\psi}^*$ for any subpath of an optimal path $\lambda_{n/t}^*$.

Proof: Let there exists a solution. Then, for any subpath of an optimal path $\lambda_{n/t}^*$ and for $1 \leq l \leq m$, the heuristic function should satisfy $h_l^*(n) \leq \psi_l$ and $h_0^*(n) \leq c_0^*$. Hence, we have $\vec{h}^*(n) \preceq \vec{\psi}^*$. Since all heuristic cost vectors are admissible, $\vec{h}(n) \preceq \vec{h}^*(n)$, and therefore, $\vec{h}(n) \preceq \vec{\psi}^*$. ■

Since all paths considered here origin from s, we have $\vec{h}(s) \preceq \vec{\psi}^*$ for any $\lambda_{s/t}^*$, and obviously, $\vec{\psi}^* \preceq \vec{\psi}$. Hence, if there exists a solution for a constrained search problem, we have $\vec{h}(s) \preceq \vec{\psi}$ which avoids any early termination in Step 1. Furthermore, CSA* terminates in Step 3 if OPEN is empty. Such a termination has to be avoided prior to CSA* finds an optimal solution for a given problem.

Lemma 1: If there exists a solution, then there exists $\xi(\lambda_{n_i/t}^*)$ on OPEN such that $\vec{f}(\lambda_{n_i/t}^*) \preceq \vec{\psi}^*$ before CSA* terminates.

Proof: Let a solution exists, then according to *Theorem* 1, we have $\vec{h}(s) \preceq \vec{\psi}^*$. Therefore, CSA* will not terminate in Step 1. Assume that $\xi(\lambda_{s/t}^*) = \{s, \vec{0}, \vec{h}(s), \text{NULL}\}$ is on OPEN. Then CSA* has not yet completed its first iteration and $\vec{f}(\lambda_s^*) = \vec{f}(\lambda_{s/t}^*) = \vec{h}(s)$. Hence, we have

$$\vec{f}(\lambda_{s/t}^*) \preceq \vec{\psi}^*,$$

where $s \equiv n_i$.

Now assume that $\xi(\lambda_{s/t}^*)$ is on CLOSED. Let $\lambda_t^* = \langle s, n_1, n_2, \ldots, n_{i-1}, n_i, \ldots, t\rangle$ and $\lambda_{n_{i-1}/t}^*$ be the longest subpath of λ_t^* on CLOSED. Since n_i is the successor of n_{i-1} on λ_t^*, whenever $\xi(\lambda_{n_{i-1}/t}^*)$ is moved from OPEN to CLOSED, CSA* must have recorded $\xi(\lambda_{n_i/t}^*)$ on OPEN and computed $\vec{g}(\lambda_{n_i/t}^*) = \vec{g}(\lambda_{n_{i-1}/t}^*) + \vec{c}(n_{i-1}, n_i)$ and $\vec{f}(\lambda_{n_i/t}^*) = \vec{g}(\lambda_{n_i/t}^*) + \vec{h}(n_i)$. Using the admissibility of $\vec{h}$, we have

$$\begin{aligned} \vec{f}(\lambda_{n_i/t}^*) &\preceq \vec{g}(\lambda_{n_i/t}^*) + \vec{h}^*(n_i), \\ f_0(\lambda_{n_i/t}^*) &\leq g_0(\lambda_{n_i/t}^*) + h_0^*(n_i). \end{aligned} \tag{9}$$

We know that $c_0^* = g_0(\lambda_t^*) = g_0(\lambda_{n_i/t}^*) + c_0(\lambda_{n_it}^*)$ and $c_0(\lambda_{n_it}^*) \geq h_0^*(n_i)$. Hence, we have

$$g_0(\lambda_{n_i/t}^*) + h_0^*(n_i) \leq c_0^*. \tag{10}$$

Combining (9) and (10) yields

$$f_0(\lambda_{n_i/t}^*) \leq c_0^*. \tag{11}$$

Since $\lambda_{n_i/t}^*$ is a subpath of the optimal path, all constraints must satisfy $\vec{f}(\lambda_{n_i/t}^*) \preceq \psi$ and therefore,

$$\vec{f}(\lambda_{n_i/t}^*) \preceq \vec{\psi}^*.$$

This confirms that the path λ_{n_i} cannot be pruned in Step 6b in Algorithm 1. Since $\lambda_{n_i/t}^*$ is a subpath of the optimal path, it cannot be dominated by any other path leading to n_i. However, there can be another path leading to n_i with same f-cost. If so, that can also be considered as a subpath of the optimal path. Therefore, one of the subpaths $\lambda_{n_i/t}^*$ will not be pruned in Step 6c. Consequently, when $\xi(\lambda_{n_{i-1}/t}^*)$ is recorded on CLOSED, there will be $\xi(\lambda_{n_i/t}^*)$ on OPEN such that $\vec{f}(\lambda_{n_i/t}^*) \preceq \vec{\psi}^*$. ■

We also prove a corollary to *Lemma* 1 for later use.

Corollary 1: Any entry $\xi(\lambda_n)$ that is selected from OPEN for expansion, should satisfy $\vec{f}(\lambda_n) \preceq \psi^*$.

Proof: Since $\xi(\lambda_n)$ is already on OPEN, the path λ_n has not been pruned in Step 6b. Thus, $\vec{f}(\lambda_n) \preceq \psi$. Since λ_n selected for expansion, its f_0 cost should be the lowest among the entries on OPEN and by the virtue of *Lemma* 1, $f_0(\lambda_n) \leq c_0^*$. Therefore, $\vec{f}(\lambda_n) \preceq \psi^*$. ■

Theorem 2: CSA* is admissible.

Proof: Here we use a sequence of contradictions to prove this theorem. Given a constrained search problem which has a solution, first assume that CSA* does not terminate. However, any best-first search algorithm that prunes cyclic paths should terminate on finite graphs [3]. Hence, CSA* should terminate as well. Now assume that CSA* terminates without a solution.

Since it is given that there exists a solution, according to *Theorem* 1, CSA* cannot terminate in Step 1. Then it can terminate without a solution only when OPEN is empty (Step 3), which contradicts with *Lemma* 1. Even if CSA* terminates with a solution, assume that CSA* terminates with a non-optimal path λ_t. Thus,

$$f_0(\lambda_t) > c_0^*. \tag{12}$$

CSA* check whether the target is reached in Step 5 after it has selected a subpath for expansion. Hence, when CSA* selects $\xi(\lambda_t)$ for expansion, $f_0(\lambda_t)$ should be the lowest f_0 value among the entries in OPEN. However, according to *Lemma* 1, there should exist at least one path on OPEN before CSA* terminates such that $\vec{f}(\lambda^*_{n_i/t}) \preceq \vec{\psi}^*$. Thus,

$$f_0(\lambda_{n_i/t}) \leq c_0^*. \tag{13}$$

Combining (12) and (13) yields

$$f_0(\lambda_{n_i/t}) < f_0(\lambda_t),$$

which contradicts with selection of $\xi(\lambda_t)$. Therefore, CSA* always terminates with an optimal solution. The theorem is proved. ■

By the virtue of *Theorem* 2, we can conclude that the admissibility of CSA* is comparable to that of A*.

Appendix B
Pruning Power of Heuristics in CSA*

In each iteration, CSA* removes an path entry from OPEN (Step 4) and its unpruned successors are recorded on OPEN (Step 6). CSA* becomes more efficient when it prunes more paths since a smaller number of paths are recorded in OPEN for expansion. Heuristics lead to two types of path pruning in CSA*. One of them is solely driven by the constrained heuristics and executed in Steps 1 and 6b in Algorithm 1. This type of path pruning occurs as a result of constraint violations. Earlier such violations are predicted, a smaller number of paths are recorded on OPEN. Second type of path pruning is driven by both major and constrained heuristics. This type of path pruning occurs as a result of dominance checks in Steps 6c and 6d in Algorithm 1. The most efficient execution of CSA* would be to record only the subpaths of the optimal path on OPEN. Such an execution would only be possible if CSA* is guided by heuristics which can always estimate the remaining cost precisely. However, such heuristic estimations are far from reality in many applications.

First we investigate the sufficient and necessary conditions for CSA* to select a path entry from OPEN for expansion. Then the path selection efficiency of CSA* is compared with different heuristics.

Definition 3: A path λ_n is said to be *ψ^*-bounded* if every subpath $\lambda_{n_i/n}$ satisfies

$$\vec{g}(\lambda_{n_i/n}) + \vec{h}(n_i) \preceq \psi^*.$$

Definition 4: A path λ_n is said to be *strictly ψ^*-bounded* if λ_n is ψ^*-bounded and every subpath $\lambda_{n_i/n}$ satisfies

$$g_0(\lambda_{n_i/n}) + h_0(n_i) < c_0^*.$$

Theorem 3: A sufficient condition for CSA* to select an entry $\xi(\lambda_n)$ for expansion is that λ_n is a nondominated and strictly ψ^*-bounded path.

Proof: We proceed by contradiction. Assume that there exists a nondominated and strictly ψ^*-bounded path λ_n and the entry $\xi(\lambda_n)$ is not yet been selected for expansion at the termination of CSA*. Since λ_n is strictly ψ^*-bounded, $\vec{f}(\lambda_{s/n}) \preceq \psi^*$ and therefore, CSA* cannot terminate in Step 1 in Algorithm 1. Now it can terminate only when a target is reached (Step 5) or OPEN is empty (Step 3). Since λ_n is strictly ψ^*-bounded, its subpaths $\lambda_{n_i/n}$ are not pruned in Step 6b. Since λ_n is nondominated, its subpaths $\lambda_{n_i/n}$ are not pruned in Steps 6c and 6d. Hence, at anytime before the termination of CSA*, there is always an entry $\xi(\lambda_{n_i/n})$ on OPEN. Therefore, OPEN cannot be empty and CSA* can terminate only when the target is reached. CSA* checks whether the target is reached only after it has selected an entry on OPEN for expansion (Step 5). When CSA* selects $\{t, \vec{g}(\lambda'_t), \vec{f}(\lambda'_t), p(\lambda'_t)\}$ for expansion, $f_0(\lambda'_t) = c_0^*$ should be the lowest f_0 value among all entries on OPEN. Since λ_n is strictly ψ^*-bounded,

$$g_0(\lambda_{n_i/n}) + h_0(n_i) < c_0^*,$$
$$f_0(\lambda_{n_i/n}) < c_0^*.$$

However, $\xi(\lambda_{n_i/n})$ is still on OPEN, which is a contradiction. Therefore, $\xi(\lambda_n)$ must be selected for expansion before CSA* terminates. ■

Theorem 4: A necessary condition for CSA* to select an entry $\xi(\lambda_n)$ for expansion is that λ_n is a ψ^*-bounded path.

Proof: If CSA* selects $\xi(\lambda_n)$ from OPEN for expansion, from *Corollary* 1, we have $\vec{f}(\lambda_n) \preceq \psi^*$. Before the time λ_n is on OPEN, all of its subpaths $\lambda_{n_i/n}$ must have been on OPEN and now they must have already been expanded. At the time $\lambda_{n_i/n}$ is expanded, it should have satisfied

$$\vec{f}(\lambda_{n_i/n}) = \vec{g}(\lambda_{n_i/n}) + \vec{h}(n_i) \preceq \psi^*.$$

Hence, the path λ_n is ψ^*-bounded. ■

Definition 5: A heuristic vector $\vec{h}$ is said to be *more informed* than another heuristic vector $\vec{h}'$ if both are admissible and satisfy

$$\vec{h}'(n) \preceq \vec{h}(n) \ \wedge \ h'_0(n) < h_0(n), \quad \forall n \in \mathcal{N} \setminus t.$$

Likewise, an algorithm using $\vec{h}$ is said to be more informed than that using $\vec{h}'$.

According to the above definition, the major heuristic cost of $\vec{h}$ must be greater than that of $\vec{h}'$ for all the nodes, except for the non-target node, for $\vec{h}$ to be considered as more informed than $\vec{h}'$. However, the rest of the heuristics are rather loosely compared.

Lemma 2: Let a heuristic vector $\vec{h}$ be more informed than another heuristic vector $\vec{h}'$. If a path λ_n is ψ^*-bounded when evaluated by CSA* with $\vec{h}$, then λ_n is strictly ψ^*-bounded when evaluated by CSA* with $\vec{h}'$.

Proof: Since λ_n is ψ^*-bounded when evaluated by CSA* with $\vec{h}$, by the virtue of *Definition* 3, every subpath $\lambda_{n_i/n}$ satisfies

$$\vec{g}(\lambda_{n_i/n}) + \vec{h}(n_i) \preceq \psi^*.$$

Since $\vec{h}$ is more informed than $\vec{h}'$, from *Definition* 5, we have $\vec{h}'(n_i) \preceq \vec{h}(n_i)$ for every non-target node n_i. Hence, we have

$$\vec{g}(\lambda_{n_i/n}) + \vec{h}'(n_i) \preceq \psi^*, \tag{14}$$

which implies that λ_n is ψ^*-bounded when evaluated by CSA* with $\vec{h}'$. Incidentally,

$$g_0(\lambda_{n_i/n}) + h_0(n_i) \leq c_0^*,$$

and we also know that $h_0'(n_i) < h_0(n_i)$ for every non-target node n_i given that $\vec{h}$ is more informed than $\vec{h}'$. Hence, we have,

$$g_0(\lambda_{n_i/n}) + h_0'(n_i) < c_0^*. \tag{15}$$

Now, (14) and (15) imply that λ_n is strictly ψ^*-bounded when evaluated by CSA* with $\vec{h}'$. ■

Theorem 5: Let algorithms CSA$_1$* and CSA$_2$* be two variations of CSA* which only differ in use of heuristics. If CSA$_1$* is more informed than CSA$_2$*, then all nondominated paths expanded by CSA$_1$* are also expanded by CSA$_2$*.

Proof: Let both algorithms CSA$_1$* and CSA$_2$* be guided by admissible heuristics $\vec{h}$ and $\vec{h}'$, respectively. Since CSA$_1$* is more informed than CSA$_2$*, from *Definition* 5, we have that $\vec{h}'(n) \preceq \vec{h}(n)$ and $h_0'(n) < h_0(n)$ for every non-target node n. If CSA$_1$* expands a nondominated path λ_n, according to *Theorem* 4, the path λ_n should be ψ^*-bounded. However, if λ_n is ψ^*-bounded when it is evaluated by CSA$_1$*, according to *Lemma* 2, λ_n is strictly ψ^*-bounded when it is evaluated by CSA$_2$*. Therefore, according to *Theorem* 3, λ_n should be expanded by CSA$_2$* as well. ■

Theorem 5 unleashes the pruning power of heuristics used in CSA* up to a certain extent. CSA* with more informed heuristics prunes more paths and as a result, it expands a smaller number of them. This is quite understandable as more informed heuristics can help in predicting violations of constraints earlier. Unfortunately, it cannot verify that every path selected by CSA* for expansion are nondominated under the assumption of admissible heuristics. Therefore, the above theorem is unable to demonstrate the pruning power of heuristics over all paths selected by CSA* for expansion.

Appendix C
Optimality of CSA* with Consistent Heuristics

In this section, we further analyze the efficiency of proposed CSA* algorithm with respect to the set of paths expanded. The number of path expansions of CSA* can be minimized by selecting only nondominated paths in Step 4, but admissible heuristics are unable to guarantee that. Hence, we introduce the concept of consistent heuristic vectors as an extension to its scalar counterpart given in [3].

Definition 6: A heuristic vector $\vec{h}$ is said to be *consistent* if it satisfies

$$\vec{h}(n) \preceq \vec{k}(n, n') + \vec{h}(n'), \quad \forall n, n' \in \mathcal{N}. \tag{16}$$

Theorem 6: All consistent heuristic vectors are admissible.

Proof: Any consistent heuristic vector $\vec{h}$ satisfies (16). By replacing n' with t, we have

$$\vec{h}(n) \preceq \vec{k}(n, t) + \vec{h}(t), \quad \forall n \in \mathcal{N}.$$

We already know that $\vec{h}^*(n) = \vec{k}(n, t)$ and $\vec{h}(t) = \vec{0}$. Incidentally,

$$\vec{h}(n) \preceq \vec{h}^*(n), \quad \forall n \in \mathcal{N}.$$

Therefore, $\vec{h}(n)$ is admissible. ■

Now with consistent heuristics, we revisit *Theorem* 4 which established a neccessary condition for CSA* to select a path entry for expansion.

Theorem 7: If the heuristics are consistent, a necessary condition for CSA* to select an entry $\xi(\lambda_n)$ for expansion is that λ_n is nondominated.

Proof: We proceed by contradiction. Assume that CSA* is guided by consistent heuristics and it expands a dominated path λ_n which is on OPEN. However, if $\xi(\lambda_n)$ is on OPEN, λ_n cannot be dominated by the already expanded paths to node n. Hence, there must be another path λ_n' which is yet to be discovered such that

$$\begin{aligned} \vec{f}(\lambda_n') &\prec \vec{f}(\lambda_n), \\ \vec{g}(\lambda_n') + \vec{h}(n) &\prec \vec{g}(\lambda_n) + \vec{h}(n), \\ \vec{g}(\lambda_n') &\prec \vec{g}(\lambda_n). \end{aligned} \tag{17}$$

Since λ_n' is nondominated, its subpaths cannot be pruned by CSA*. Therefore, by the time that CSA* expands λ_n, a subpath $\lambda_{n_i/n}'$ must be on OPEN. We can decompose the cost of the nondominated path λ_n' as

$$\vec{g}(\lambda_n') = \vec{g}(\lambda_{n_i/n}') + \vec{c}(\lambda_{n_i n}'). \tag{18}$$

Obviously, $\vec{k}(n_i, n) \preceq \vec{c}(\lambda_{n_i n}')$. Therefore, using (16), we can obtain

$$\vec{h}(n_i) \preceq \vec{c}(\lambda_{n_i n}') + \vec{h}(n),$$

in which $\vec{c}(\lambda_{n_i n}')$ can be replaced using (18) such that

$$\vec{g}(\lambda_{n_i/n}') + \vec{h}(n_i) \preceq \vec{g}(\lambda_n') + \vec{h}(n). \tag{19}$$

Combining (17) and (19) yields that

$$\begin{aligned} \vec{g}(\lambda_{n_i/n}') + \vec{h}(n_i) &\prec \vec{g}(\lambda_n) + \vec{h}(n), \\ \vec{f}(\lambda_{n_i/n}') &\prec \vec{f}(\lambda_n), \qquad (20) \\ f_0(\lambda_{n_i/n}') &\leq f_0(\lambda_n). \qquad (21) \end{aligned}$$

In accordance with Step 4 in Algorithm 1, (20) and (21) verify that $\lambda_{n_i/n}'$ must be expanded before expanding λ_n. This is valid for all the subpaths of λ_n'. By the time $\xi(\lambda_n')$ is recorded on OPEN, $\xi(\lambda_n)$ has to be removed from OPEN as λ_n is dominated by λ_n'. Hence, λ_n cannot be expanded before CSA* terminates. This contradicts with our initial assumption, thus, the proof is completed. ■

The above theorem draws an important inference that all path entries selected by CSA* for expansion are nondominated when it is guided by consistent heuristics. Hence, once a path is recorded on CLOSED (Step 4), it cannot be dominated by any other path leading to the same node.

Definition 7: An algorithm CSA$_1$ is said to *dominate* an algorithm CSA$_2$ if every path expanded by CSA$_1$ is also expanded by CSA$_2$. CSA$_1$ is said to *strictly dominate* CSA$_2$ if CSA$_1$ dominates CSA$_2$ and CSA$_2$ does not dominate CSA$_1$.

Theorem 8: Let algorithms CSA$_1$* and CSA$_2$* be two variations of CSA* guided by a consistent heuristic vector $\vec{h}$ and an admissible heuristic vector $\vec{h}'$ (not necessarily consistent), respectively. If CSA$_1$* is more informed than CSA$_2$*, then CSA$_1$* dominates CSA$_2$*.

Proof: Since CSA$_1$* is more informed than CSA$_2$*, according to *Theorem* 5, all nondominated paths expanded by CSA$_1$* are also expanded by CSA$_2$*. Since CSA$_1$* is guided by a consistent heuristic function, according to *Theorem* 7, all paths expanded by CSA$_1$* are nondominated. Hence, all the paths expanded by CSA$_1$* should also be expanded by CSA$_2$* and according to *definition* 7, CSA$_1$* dominates CSA$_2$*. ■

Corollary 2: Let algorithms CSA$_1$* and CSA$_2$* be two variations of CSA* guided by consistent heuristics $\vec{h}$ and $\vec{h}'$, respectively. If CSA$_1$* is more informed than CSA$_2$*, then CSA$_1$* dominates CSA$_2$*.

Proof: According to *Theorem* 6, all consistent heuristic vectors are admissible. The proof of *Theorem* 8 does not require $\vec{h}'$ to be consistent, but admissible. Therefore, this corollary trivially follows from *Theorem* 8. ■

Definition 8: An algorithm is said to be *optimal* over a class of algorithms if it dominates all algorithms in that class [3].

Theorem 9: Let CSA* be guided by consistent heuristics and $\mathcal{C}_a$ be a class of admissible constrained search algorithms which are no more informed than CSA*. Then CSA* is optimal over $\mathcal{C}_a$.

Proof: If CSA* is optimal over $\mathcal{C}_a$, it should dominate any algorithm, say CSA, in $\mathcal{C}_a$. Assume the contrary. Then CSA finds an optimal path from a source node s to a target node t on a given digraph $\mathcal{G}$ without exploring all paths that are explored by CSA* to find an optimal solution. Let λ_n be one such path of which $\vec{f}(\lambda_n) = [f_0(\lambda_n), f_1(\lambda_n), \ldots, f_i(\lambda_n), \ldots, f_m(\lambda_n)]$. Since λ_n is explored by CSA*, by *Theorem* 4, $\vec{f}(\lambda_n) \preceq \vec{\psi}^*$. Hence, we have

$$f_0(\lambda_n) \leq c_0^*.$$

Since CSA is no more informed than CSA*, it cannot identify possible violations of constraints prior to CSA* does. Hence, the only possible scenario for CSA not to explore λ_n is that there should be another path λ'_n to n such that

$$g_0(\lambda'_n) < g_0(\lambda_n), \tag{22}$$

which satisfies all the constraints. Now let $\vec{f}(\lambda'_n) = [f_0(\lambda'_n), f_1(\lambda'_n), \ldots, f_i(\lambda'_n), \ldots, f_m(\lambda'_n)]$. CSA* is guided by consistent heuristics, thus by the virtue of *Theorem* 7, λ_n must be a nondominated path to a node n. From (22), we have $f_0(\lambda'_n) < f_0(\lambda_n)$. Therefore, for λ_n to be nondominated, there exists an $i \in [1, m]$ such that

$$f_i(\lambda_n) < f_i(\lambda'_n).$$

Since CSA is admissible, it must be able to find an optimal solution for any given problem from a class of constrained search problems that have solutions. Now assume that there is another constrained search problem which is identical to the previously considered one, but with different constraints. The new constraint vector is given by $\vec{\widetilde{\psi}}^* = [\widetilde{c_0}^*, \widetilde{\psi}_1, \ldots, \widetilde{\psi}_i, \ldots, \widetilde{\psi}_m]$ such that $\widetilde{c_0}^* > c_0^*$, $\vec{f}(\lambda_n) \preceq \vec{\widetilde{\psi}}^*$, and $f_i(\lambda_n) < \widetilde{\psi}_i < f_i(\lambda'_n)$. Now assume that the only optimal path form s to t goes through n. Since s and t remain unchanged on $\mathcal{G}$, path costs and heuristic costs remain unchanged as well. This lets CSA* and CSA to behave similar to as they did in the previous search problem. Hence, CSA* should be able to find the new optimal path by expanding λ_n which satisfies the constraints. Since CSA skips expanding λ_n, it cannot find the optimal path for this particular problem which actually has a solution. Thus CSA violates its admissibility, which is contrary to our assumption. Therefore, the theorem is proven. ■

According to *Theorem* 9, any constrained search algorithm, which is no more informed than CSA*, cannot skip any path selected by CSA* with consistent heuristics in a search of an optimal solution without compromising its admissibility. Hence, CSA*, when guided by consistent heuristics, is optimal with respect to the set of paths expanded over a class of no more informed admissible constrained search algorithms.