A new polynomial time heuristic for Hamiltonian Cycle Problem

BY

YASH VINAYVANSHI

ABSTRACT

This paper describes a deterministic polynomial time heuristic for determining a hamiltonian cycle in a general undirected graph. The algorithm finds atomic cycles in the graph and integrate them to form larger cycles based on described criteria, which eventually can be as large as the hamiltonian cycles in graphs which are hamiltonian. To achieve an optimal sequence of mergers, the heuristic takes local decisions of merging cycles such that they do not render any vertex unreachable in future.

1. INTRODUCTION

The hamiltonian cycle problem (HCP) is, g*iven a graph G, detect and return if there exists a* simple cycle which spans all the vertices of G, called hamiltonian cycle (HC). A graph which contains at least one HC, is called hamiltonian and non hamiltonian otherwise. HCP is popular and well studied NP complete problem and has a close association to the travelling salesman problem (TSP), which is a popular optimisation problem. TSP is generally formulated as, to find the optimal cost hamiltonian cycle in a weighted complete graph which contains a search space of O(n!) HCs. However in HCP, the problem is to detect if there exist an HC in any given, sparse or dense graph G. HCP can be formulated as TSP by providing weight 1 to edges which exist in G of order n in Kn and weight greater than n to all other edges on Kn which does not exist in G, such that minimum cost of HC if it exists is n. A graph can have multiple HCs, sparse graphs generally have less number HCs while dense graphs generally contain more HCs, although there are attempts to devise some necessary and sufficient conditions for existence of HC, as given by[], for instance, based on maximum degree of graph, however, a large number of sparse graphs does not fall under this condition.
The exact algorithm for hamiltonian cycle detection requires exploration of entire search space of simple paths in a graph which is exponential for most graphs and grows inordinately in time with number of vertices. Although some optimisation can be achieved by skipping unfeasible part of search spaces earlier, or casting HC as a TSP problem and apply dynamic programming, the best exact methods are still exponential, if not factorial in time and space. NP problems cannot be solved in polynomial time until some inherent structure is discovered in such problems which can be exploited to avoid searching. There is no mathematical construct that brings a necessary and sufficient condition for this problem on general graphs and solve HCP in polynomial time for any general graphs. However, there are several polynomial time algorithms proposed for detection of HCs in special graphs which have a greatly reduced search space, or probabilistic algorithms whose worst time complexities are still exponential. Solving this problem by exhaustively checking necessary conditions as given by Ore etc is also exponential. This brings attempts in heuristics which are not guaranteed to always give accurate result for HCP, but several effective polynomial time heuristics are produced, which solve HCP with fair accuracy even for very large graphs, some of which are Lin Karnighan, snake and ladder, which utilise k-opt transformations or k-change transformations. These iterate over entire graph every time. The heuristics presented in this paper have a different approach where it tries to discover hamiltonian cycle by incrementally expanding a seed cycle, until it achieves size n. However this heuristic is takes decisions based more on local picture than global picture as in published heuristics. The way to incorporate global picture of graph in local decision making is however still under investigation.
The merge cycle heuristic (MCH) for HCP discussed in this paper discovers atomic cycles (smallest indivisible cycles) in a graph and merges them based on certain criteria described later to discover a larger cycle, called the grand cycle in the graph which might reach to the size of HC subsequently. In a hamiltonian graph, out of all possible merger sequences of atomic cycles, only several may lead to a HC, others may reach to a situation where further expansion of grand cycle, which has yet not reached size n, is not possible.

The heuristics first discovers all atomic cycles in a graph G and builds a cycle adjacency graph G', described later, as steps of preprocessing G. And based on G', the heuristic attempts to predict in which direction to expand the grand cycle so that the probability of it reaching to one of the HCs increases. With each iteration of expanding the grand cycle, the heuristic updates the cycles adjacency graph and some data associated to it. The heuristics chooses to expand the grand cycle such that no neighbouring vertex is rendered unincludable in future.
The organisation of paper is as follows, Section 2 contains some preliminary mathematical definitions and establishes notation to be used in paper later. In Section 3, we present the abstract idea, some theorems and intuitions about the algorithm we are proposing. In Section 4, we state the algorithm for determining HC by integrating cycles with k look ahead, & its limitation to establish the background for improvements introduced by the heuristics. In section 5, we present the heuristics and its complexity. In section 6, we present the computational results of given heuristic. Section 7 contains some further questions and scopes regarding concepts introduced in this paper. To make the paper lucid, A variety of illustrations are included.

2. DEFINTITONS & THEOREMS

Definition 2.1. (Graph) *A graph is a two tuple $G = (V, E)$, where V is a set of symbols representing vertices and E is a set of pairs $(u \in V, v \in V)$ and represent edges between a pair of vertices u, v. If (u, v) is an ordered pair, then it represents a directed edge from u to v, otherwise if unordered, (u, v) represents an undirected edge. A graph with all edges being undirected is called an undirected graph. An undirected edge can be modelled as two directed edges in opposite directions. In a simple graph, E is not a multiset and thus does not allow duplicate edges or multi edges, and there is no edge of type <u, u> called self loop. In this context, Graph G is assumed to be simple an undirected.*

Definition2.2 (Interface) *Consider two cycle graphs $A = (V_A, E_A)$ and $B = (V_B, E_B)$. An interface between A and B is common simple path between A and B. The size of interface is the length of its path. In Fig1, cycles A & B share 3 interfaces $I1:V_1V_2V_3V_4$, $I2 : V_5V_6$ & $I3 : V_7$ of sizes 3, 1 and 0.*

Lemma 2.3. *Consider two cycle graphs $A = (V_A, E_A)$ and $B = (V_B, E_B)$. A graph $G = (V_G, E_G)$ where $V_G = V_A \cup V_B$ and $E_G = E_A \cup E_B$ If A and B contain no interface, G is disconnected and thus non hamiltonian.*

1. *If A and B contain an interface of size 0, G is non hamiltonian. (ref to Fig. 2a)*
2. *If A and B contain one interface of size 1, say, I=VV' ie such that $E_A \cap E_B = \{<V, V'>\}$ and $V_A \cap V_B = \{V, V'\}$, then G contains a hamiltonian cycle given by $H = (V_G, E_G - \{<V, V'>\})$. (ref to Fig. 2b, 2c)*
3. *If A and B contain multiple interfaces of size 1, then G will contain hamiltonian cycle only if G is non planar. (ref to Fig 2d)*
4. *If A and B contain an interface of size > 1, then G is a theta graph and thus does not contain HC (refer to Fig. 2e)*

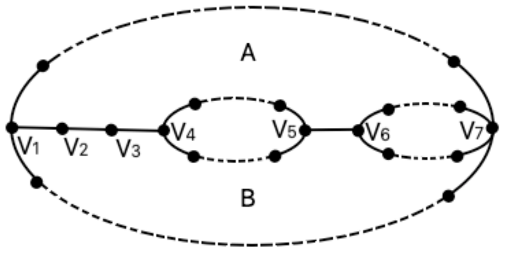


Fig. 1 Cycles A & B sharing three interfaces.

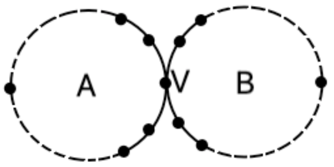


Fig. 2a Cycles A, B sharing interface of size 0.

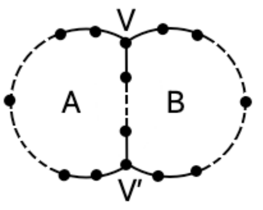


Fig. 2e Cycles A, B sharing interface of size > 1

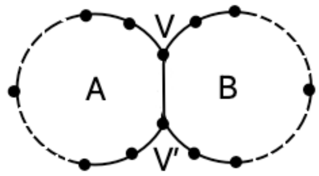


Fig. 2b Cycles A, B sharing interface of size 1, VV'

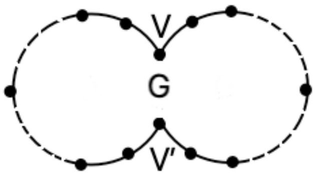


Fig. 2c Cycles A, B merged to form a cycle by deleting VV'

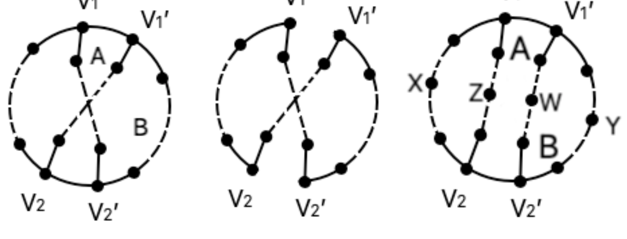


Fig 2d

[illegible]

[illegible] [illegible]

[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]

[illegible] [illegible]

[illegible]

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

3. ALGORITHM TO FIND ALL ATOMIC CYCLES IN A GRAPH

3.1 The Algorithm

[illegible]

[illegible]

```
Algorithm [illegible]
1. [illegible]
2. [illegible]
3. [illegible]
4. [illegible]
5. [illegible]
    1. [illegible]
    2. [illegible]
    3. [illegible]
        1. [illegible]
    4. [illegible]
6. [illegible]
7. [illegible]
8. [illegible]
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	None	None
[illegible]	[illegible]	[illegible]	None	None
[illegible]	[illegible]	[illegible]	None	None
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

1. [illegible]
2. [illegible]

[illegible]

[illegible]

[illegible]

```
Algorithm [illegible]
Input: [illegible]
Output: [illegible]

1. [illegible]
2. [illegible]
    1. [illegible]
    2. [illegible]
```

[illegible]

```
Algorithm [illegible]
Input: [illegible]
Output: [illegible]
```

[illegible]

```
[illegible]
```

[illegible]

```
[illegible]
```

[illegible]

```
[illegible]
```

[illegible]

Fig. 9c

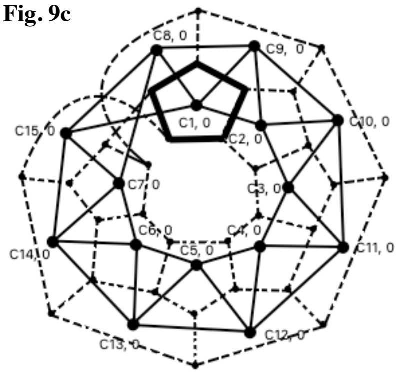


Fig. 9d

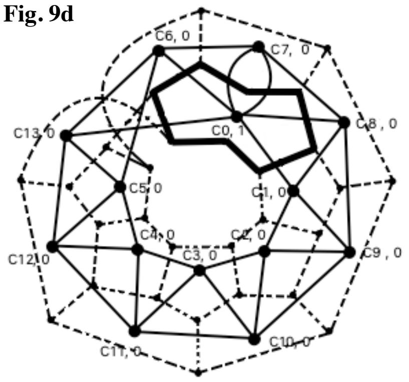


Fig. 9e

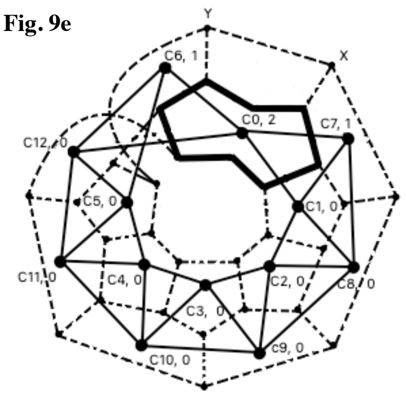


Fig. 9f

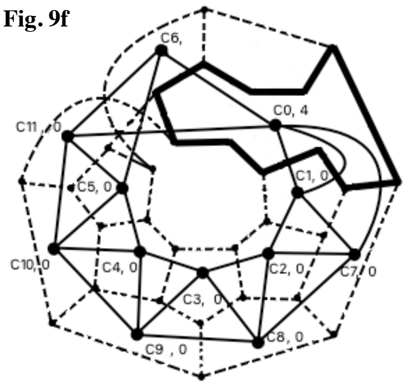


Fig. 9g

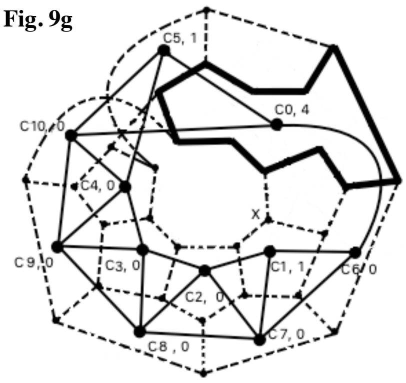


Fig. 9h

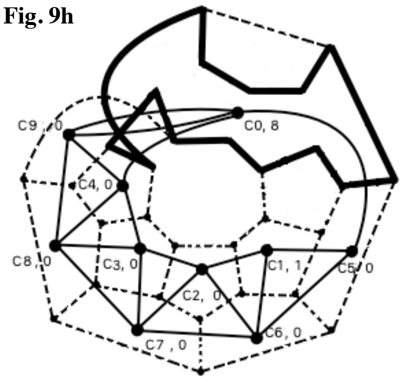


Fig. 9i

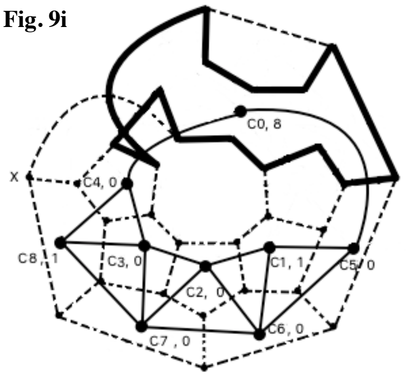


Fig. 9j

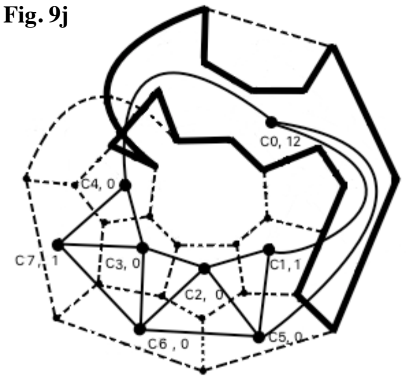


Fig. 9k

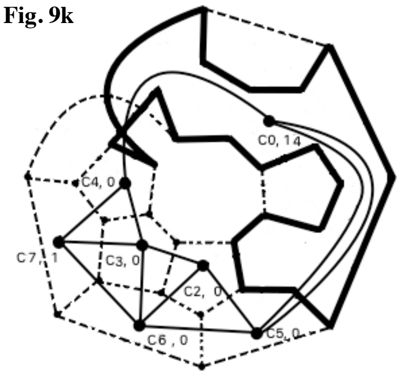


Fig. 9l

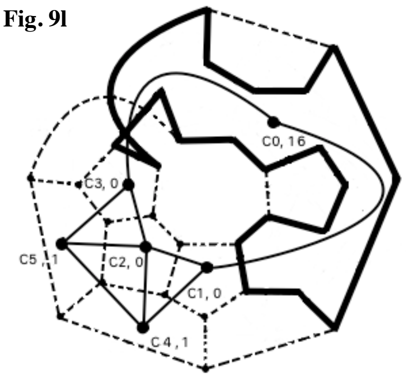


Fig. 9m

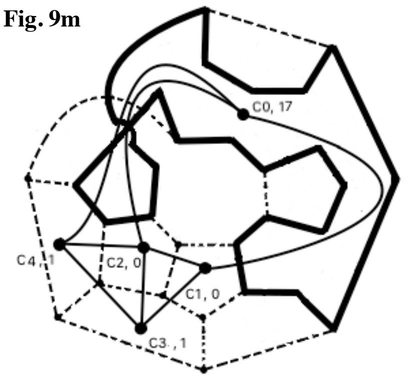


Fig. 9n

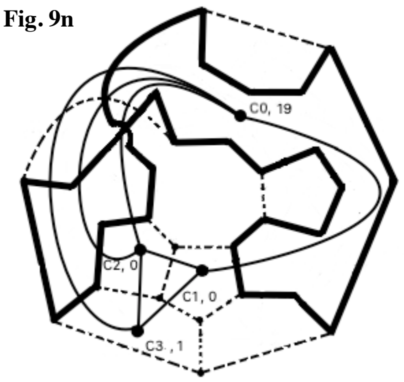


Fig. 9o

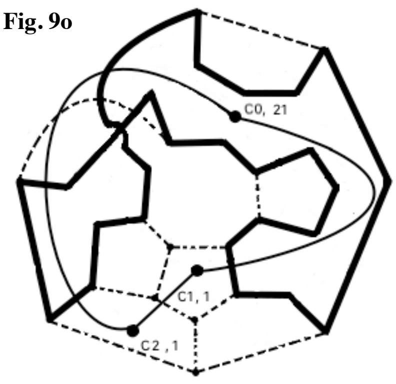


Fig. 9p

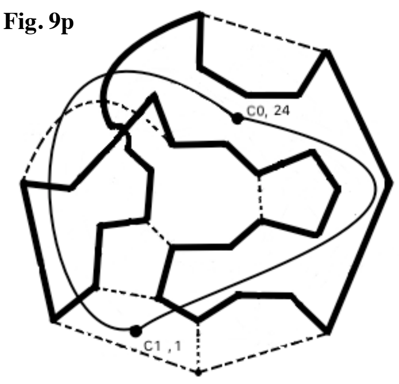


Fig. 9q

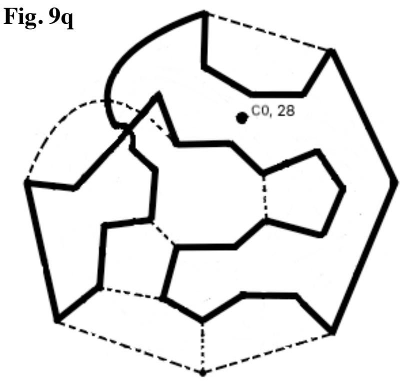
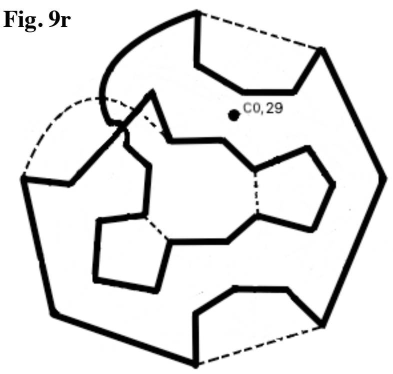


Fig. 9r



The Graph G as given **Fig. 9a** has 29 vertices, 45 edges. The find FIND_ALL_ATOMIC_CYCLES(G) method discovers 15 atomic cycles S = {C0..C15} as in **Fig. 9b**. The BUILD_CYCLE_ ADJACENCY_GRAPH(S), returns the cycle adjacency graph G' and CALCULATE_TAGS(S), returns tags for each cycle in G', as in **Fig. 9c**, the dashed lines represent G while solid lines represent G', each vertex is mapped to one cycle, and has a tag representing vertices unique to that cycle for eg C0 : V1V2V16V15V14V1 has no vertex unique to itself like V14 in C0 is shared with C8 and C15. **Fig. 9d** Since all cycles in G' has tag 0 at this point, we can choose any cycle as seed, here we have chosen C0 as seed and initialised grand cycle with it. The bold lines represent the grand cycle at a given iteration. C0 has neighbours C1, C7, C8, C9 with tags 0, 0, 0, 0 respectively as indicated by edges in G', all these neighbours have tags = 0, hence we can choose any one of these to merge into grand cycle which is C0 now, here we have chosen C1 and thus by deleting edge<V2, V16> we obtain a larger grandcycle represented by C0 now in cycle adjacency graph. **Fig. 9e** on updating the CAG, we find that C7 shared multi edge with grand cycle (C0) and so we delete C7 from R as in **Fig. 9f ,** this leads to vertices X and Y as indicated, to be unique for cycles C6 and C7 respectively, and thus, their tags are updated to 1 from 0. Now from grand cycle C0, we have four neighbours C1, C6, C7, C12 with tags 0, 1, 1, 0, thus we prefer to expand towards C6 or C7 (with labels > 0) first instead of C1 or C12. We choose to expand grand cycle towards C7 leading to a cycle adjacency graph as given in **Fig. 9g ,** again we see a multi edge from grand cycle to C1**,** thus C1 is deleted from R and vertices are reassigned names. **Fig. 9f,** the tag of cycle C1 is now 1 due to vertex indicated by X being free due to deletion of earlier cycle C1.

7. Computation, Optimization & Analysis

This section describes the computing specific part, ie data structures and methods that may be used . These methods establish the ground for analysis of this algorithm.

In terms of cycle adjacency matrix

Method	**Calculation**	**Time**		
		Worst	**Best**	**Avg**

[illegible]

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]				
		[illegible]	[illegible]	[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	[illegible]	
				[illegible]	[illegible]					
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]				[illegible]				
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]			

[illegible]

[illegible]	[illegible]	[illegible]	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]					
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		[illegible]	
[illegible]	[illegible]					[illegible]	[illegible]		[illegible]	
[illegible]	[illegible]					[illegible]	[illegible]		[illegible]	
[illegible]	[illegible]					[illegible]	[illegible]		[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	

[illegible]

[illegible]

[illegible]

[illegible]