CS60045 Artificial Intelligence Autumn 2023

Informed Search Techniques

Generic search algorithm (Best-first search)

```
Create an empty frontier, and insert start_node to frontier.
Create an empty reached table, and insert (start_node, initial_cost) to reached.
While frontier is not empty, repeat:
    Extract the best node from frontier.
    If node is a goal node, return node.
    Expand node to get the list of child nodes.
    For each child, repeat:
        Calculate new_path_cost from root to child (node_path_cost + transition_cost)
        If (child in not in reached)
            Add child to frontier.
            Record node as the parent of child.
            Add (child, new_path_cost) to reached.
        Else if (new_path_cost to child is better than the earlier path cost to child)
            Add child to frontier. (If child is already present in frontier, update the entry.)
            Record node as the parent of child.
            Replace (child, old_path_cost) by (child, new_path_cost) in reached.
Return failure.
```

Informed / Heuristic search

Let n be an open node.

$g(n)$ = cost from the root to n (not necessarily the path cost).

$h(n)$ = a heuristic estimate of the cheapest cost to reach the/a goal from n.
(We assume that $h(n) = 0$ for an (optimal) goal node.)

$f(n)$ involves $h(n)$ for expanding open nodes.

The open node with the smallest $f(n)$ value is expanded first.

The estimate $h(n)$ requires some domain-specific knowledge.

Good heuristics can improve the performance of the search significantly.

Greedy Best-First Search (GBFS)

Take $f(n) = h(n)$.

Focus is only on the rest of the search: $h(n)$.

Motivation: Try to reach the solution from the current state as quickly as possible. The open node with the smallest $h(n)$ value is apparently closest to the goal. This is a greedy move.

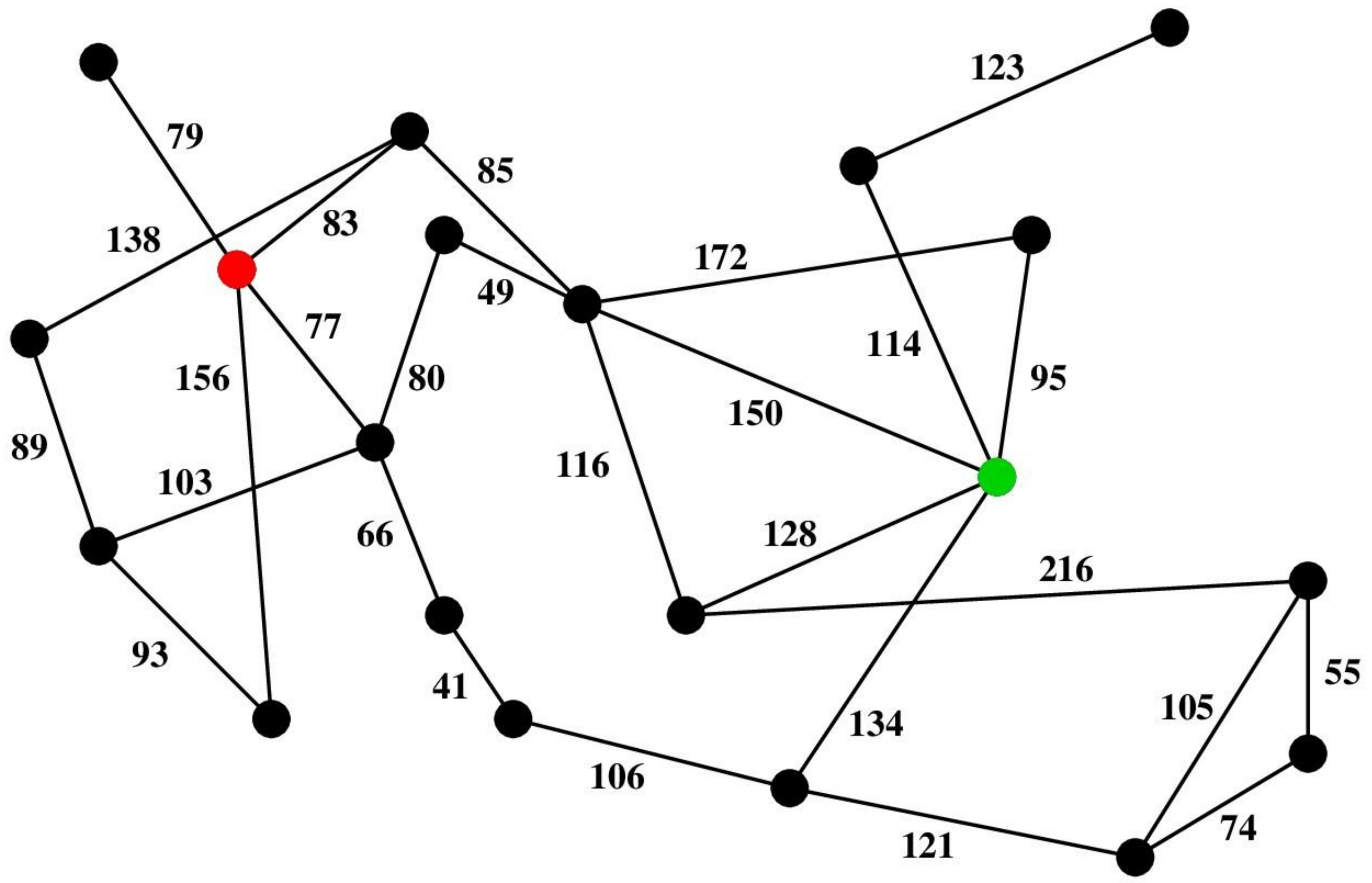
Completeness: Yes on finite state-space graphs, not necessarily otherwise.

Cost-optimality: Not guaranteed (see the example to follow).

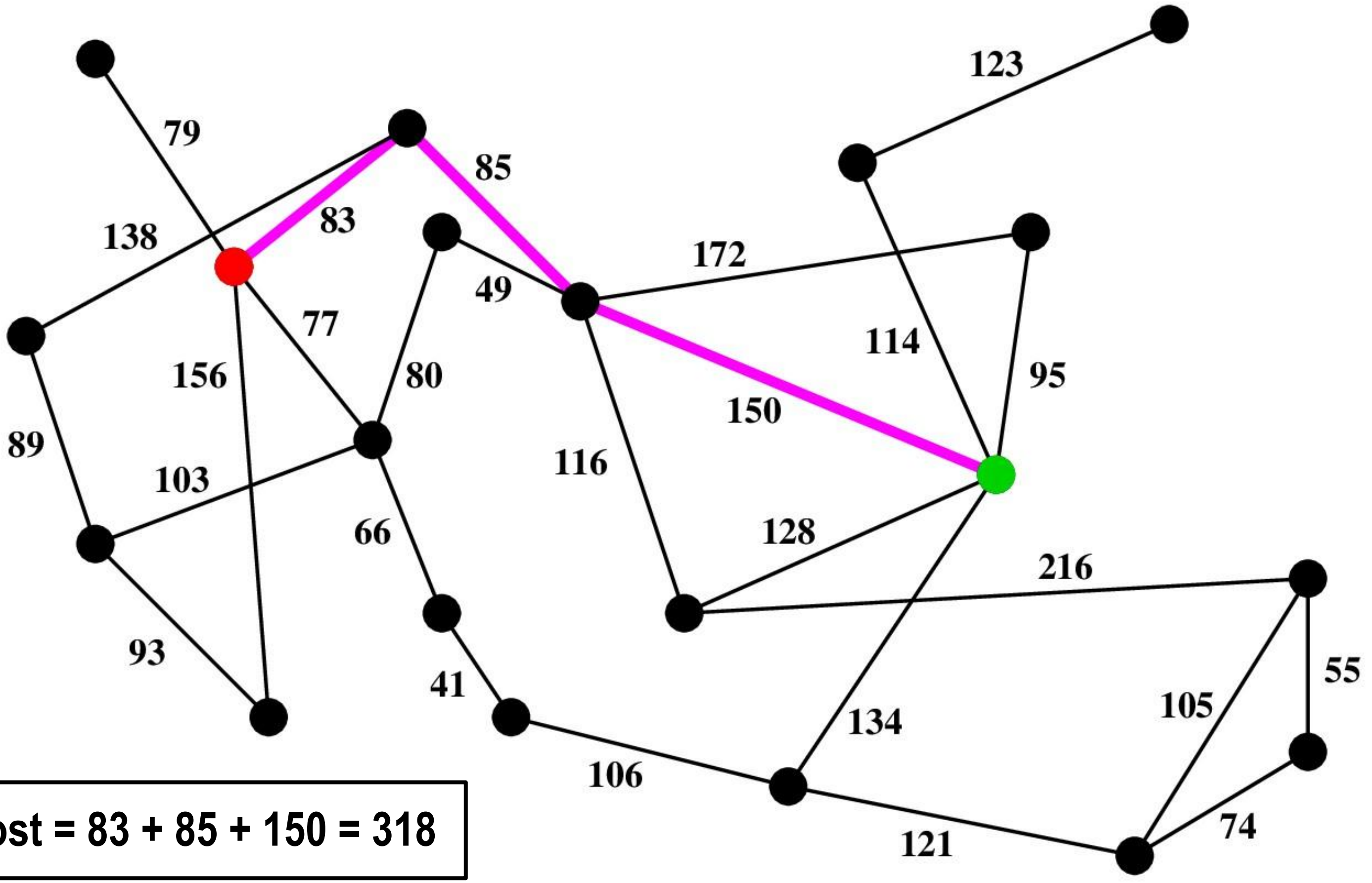
Space complexity is $O(|V|)$, where V is the vertex set of the state-space graph.

Time complexity can be $O(|V| \log |V|)$.

A path-finding example [similar to the Romania example of AIMA]

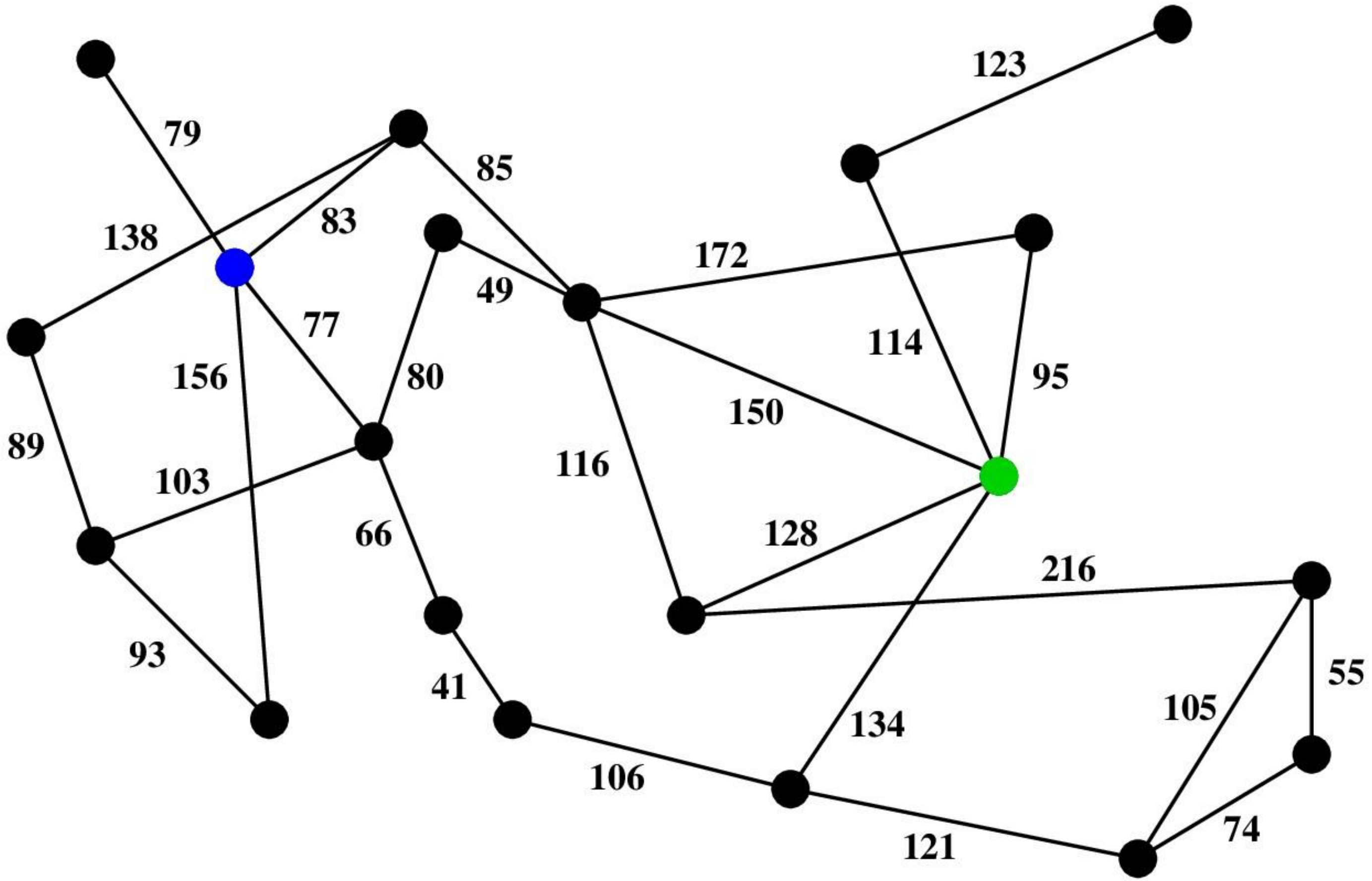


A path-finding example: The solution

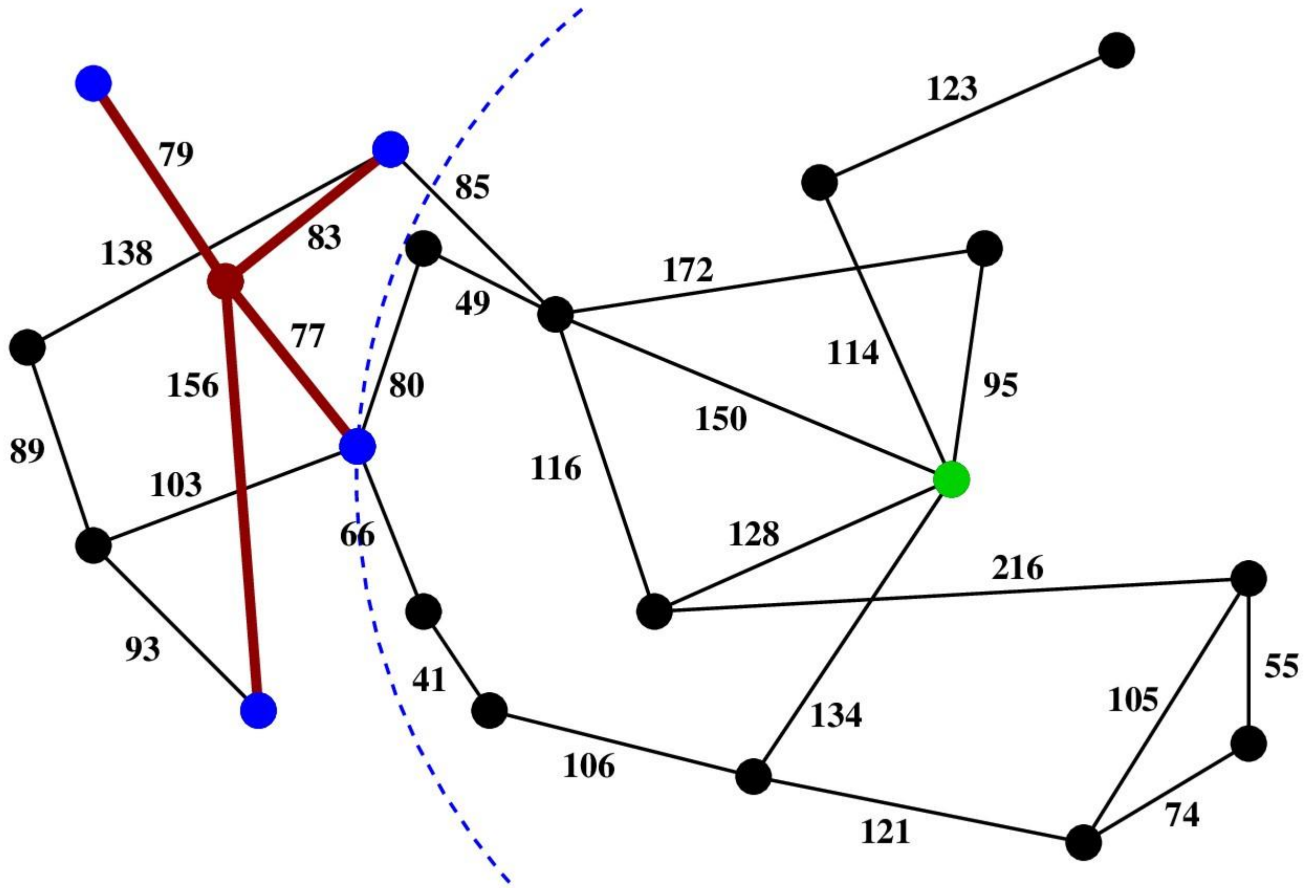


Optimal cost = 83 + 85 + 150 = 318

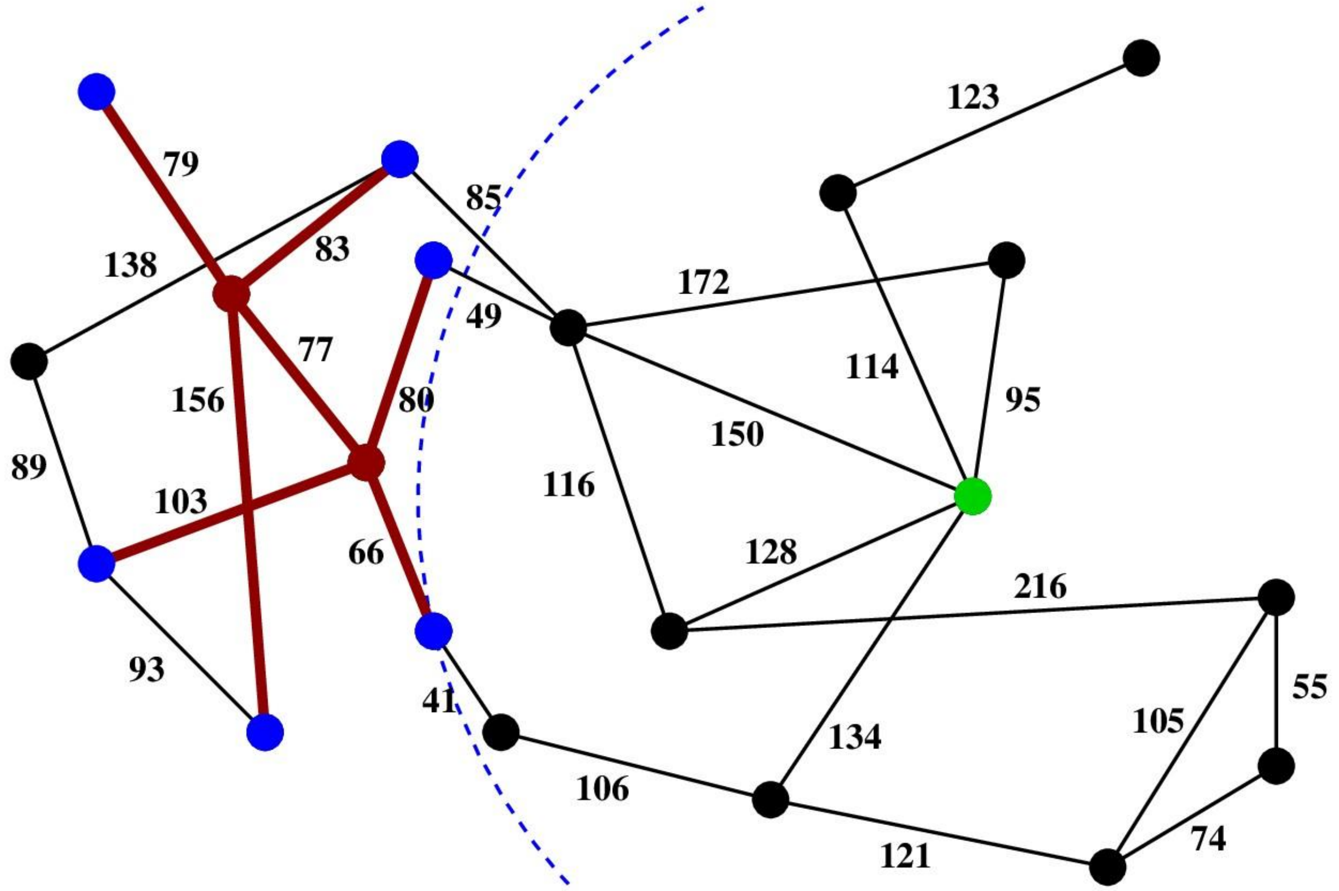
Example of GBFS (Greed does not necessarily pay off)



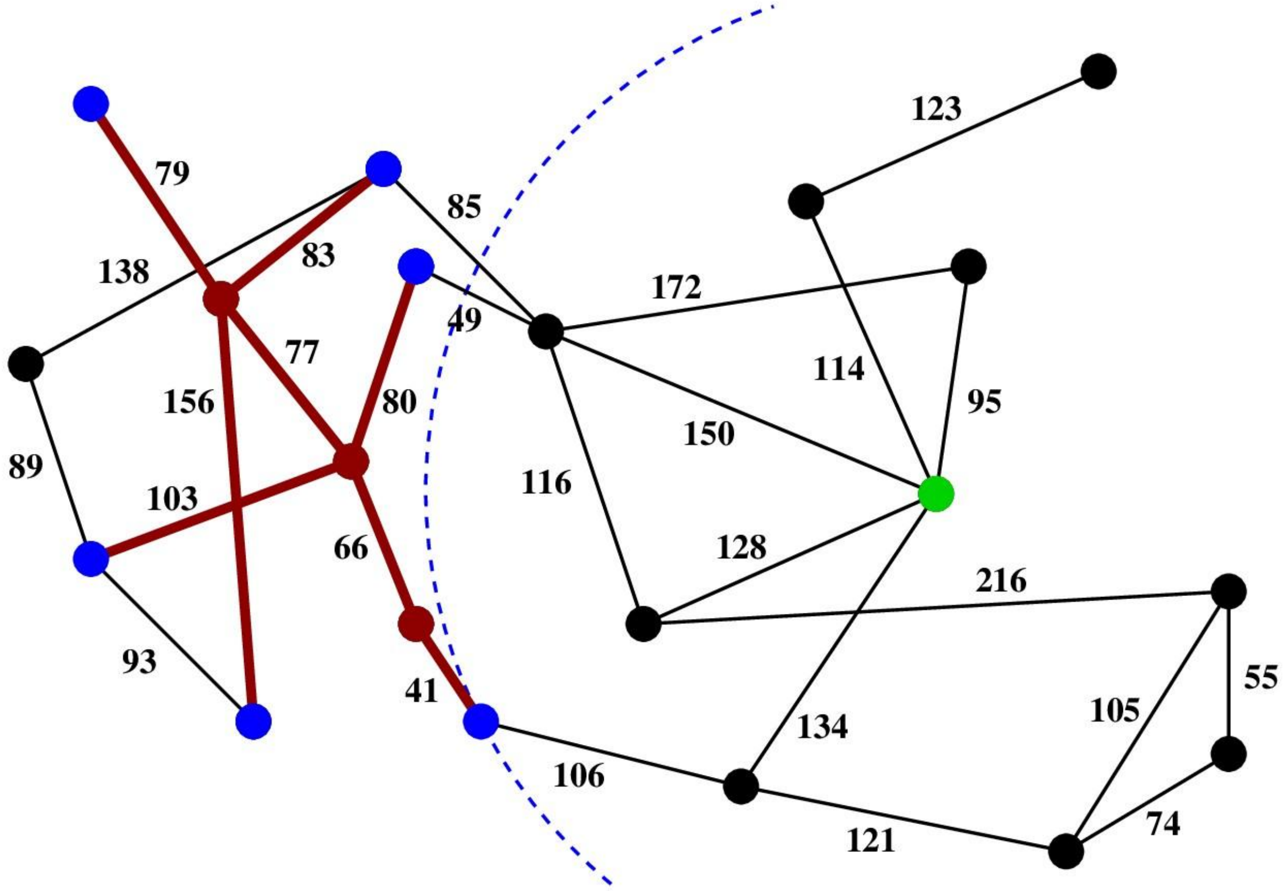
Example of GBFS



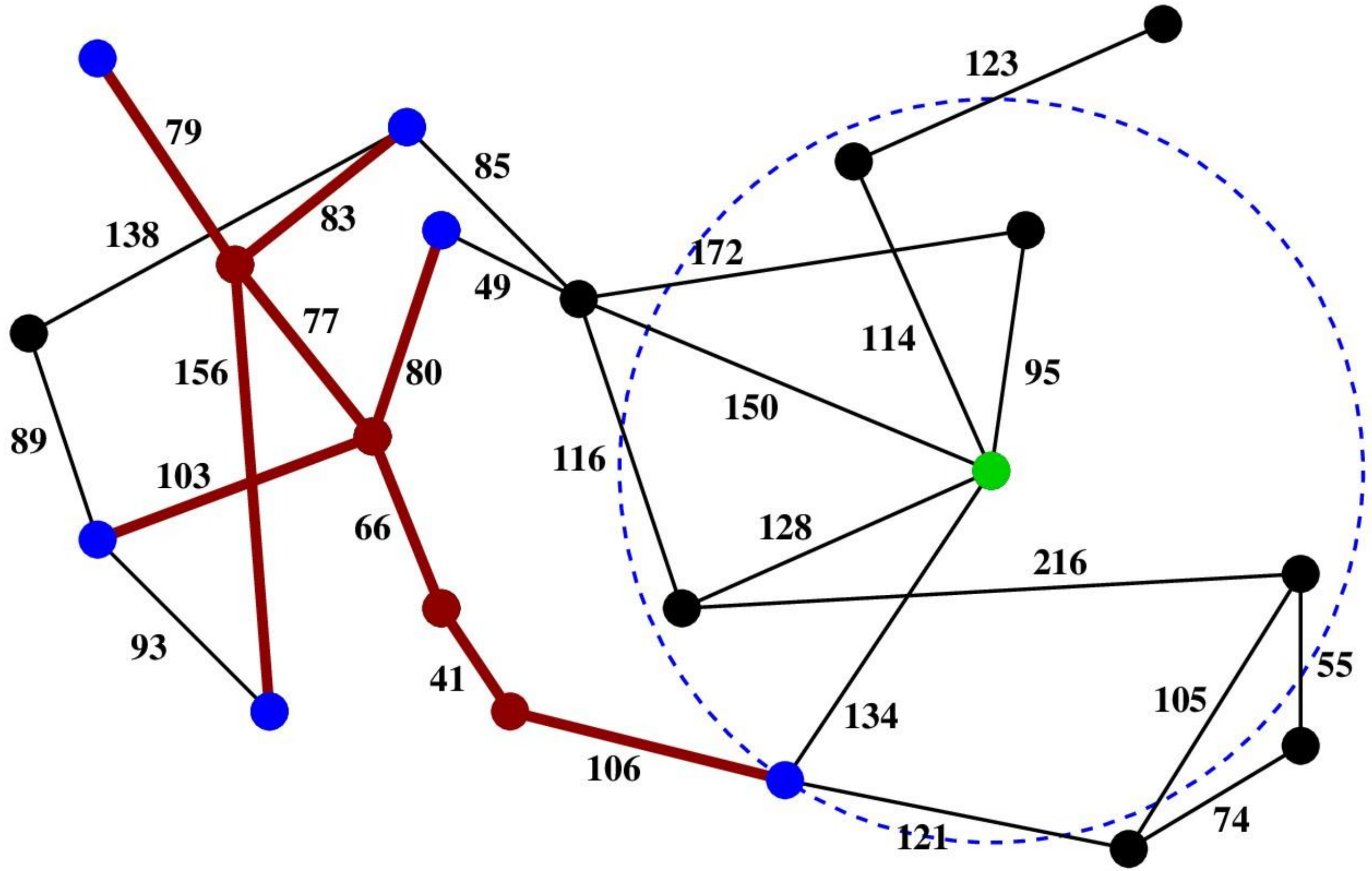
Example of GBFS



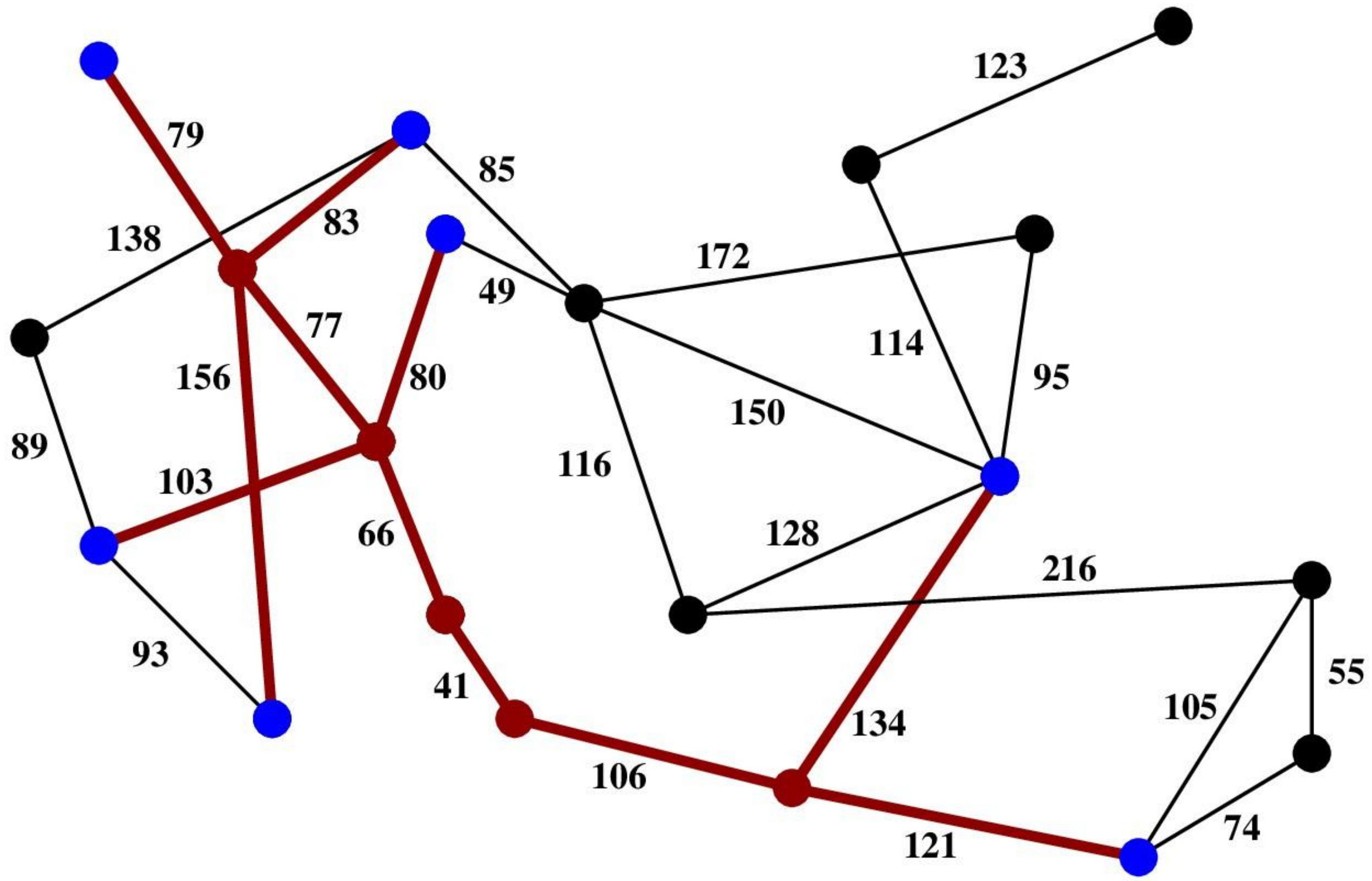
Example of GBFS



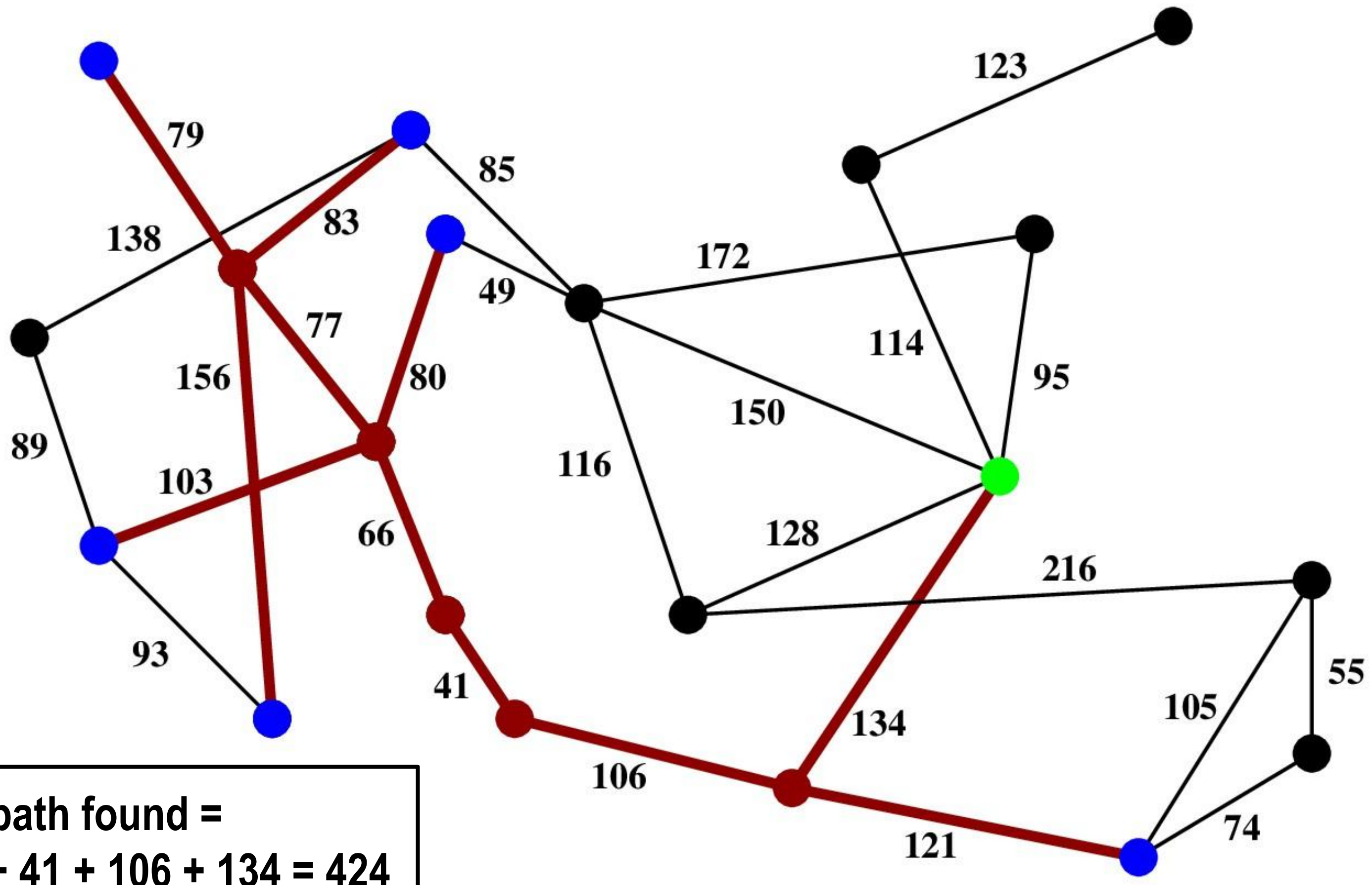
Example of GBFS



Example of GBFS



Example of GBFS



Cost of path found =
77 + 66 + 41 + 106 + 134 = 424

A* Search

Let n be an open node.

Take $g(n)$ = the path cost from the root to n, found so far.

$h(n)$ = A heuristic estimate of the cheapest way to reach the goal from n.

Take $f(n) = g(n) + h(n)$.

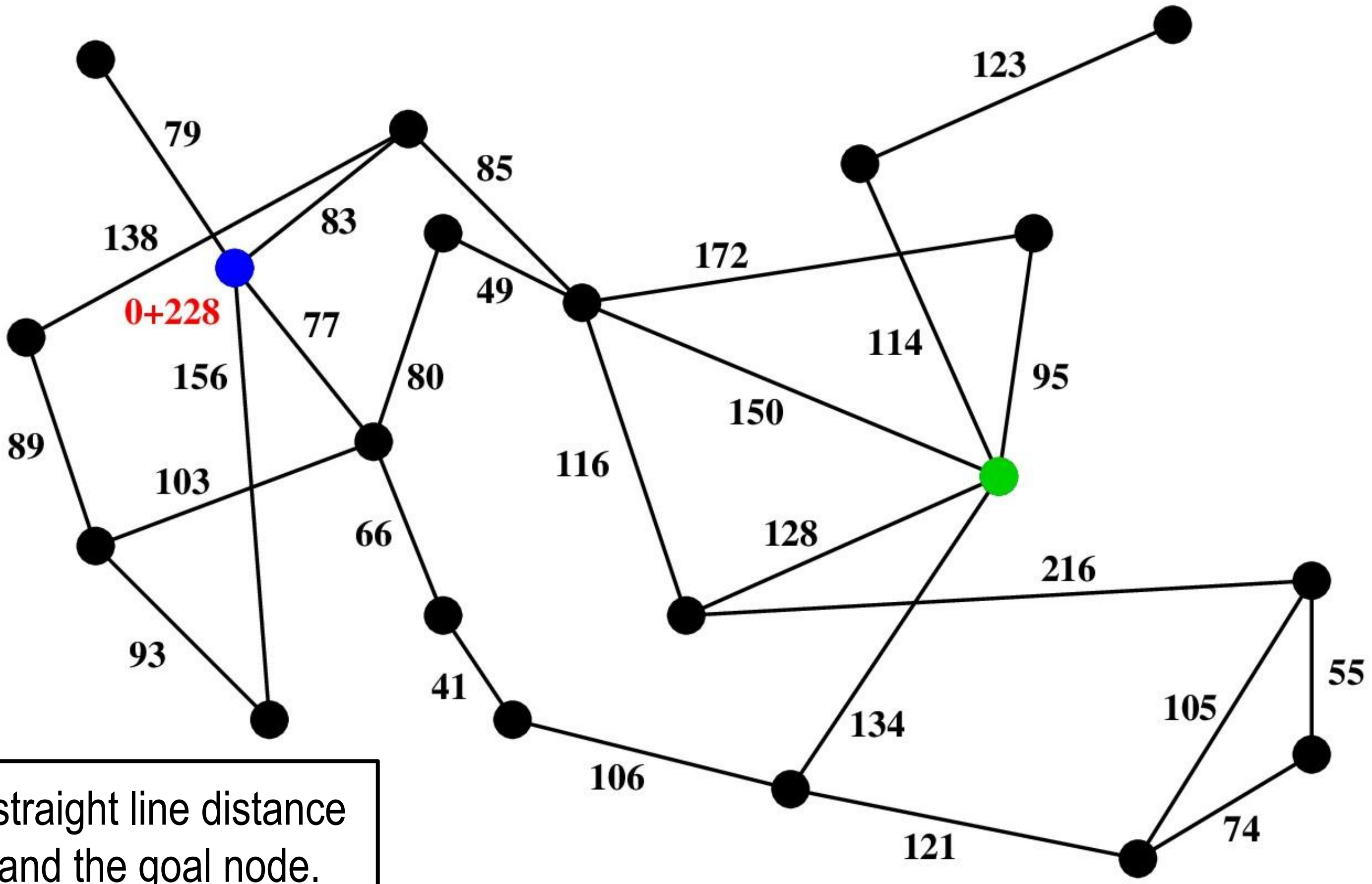
Motivation: Find a cheapest path from the root to the goal via n.

If we take $h(n) = 0$, we get the uniform-cost search which is cost-optimal.

A* is complete for finite state spaces.

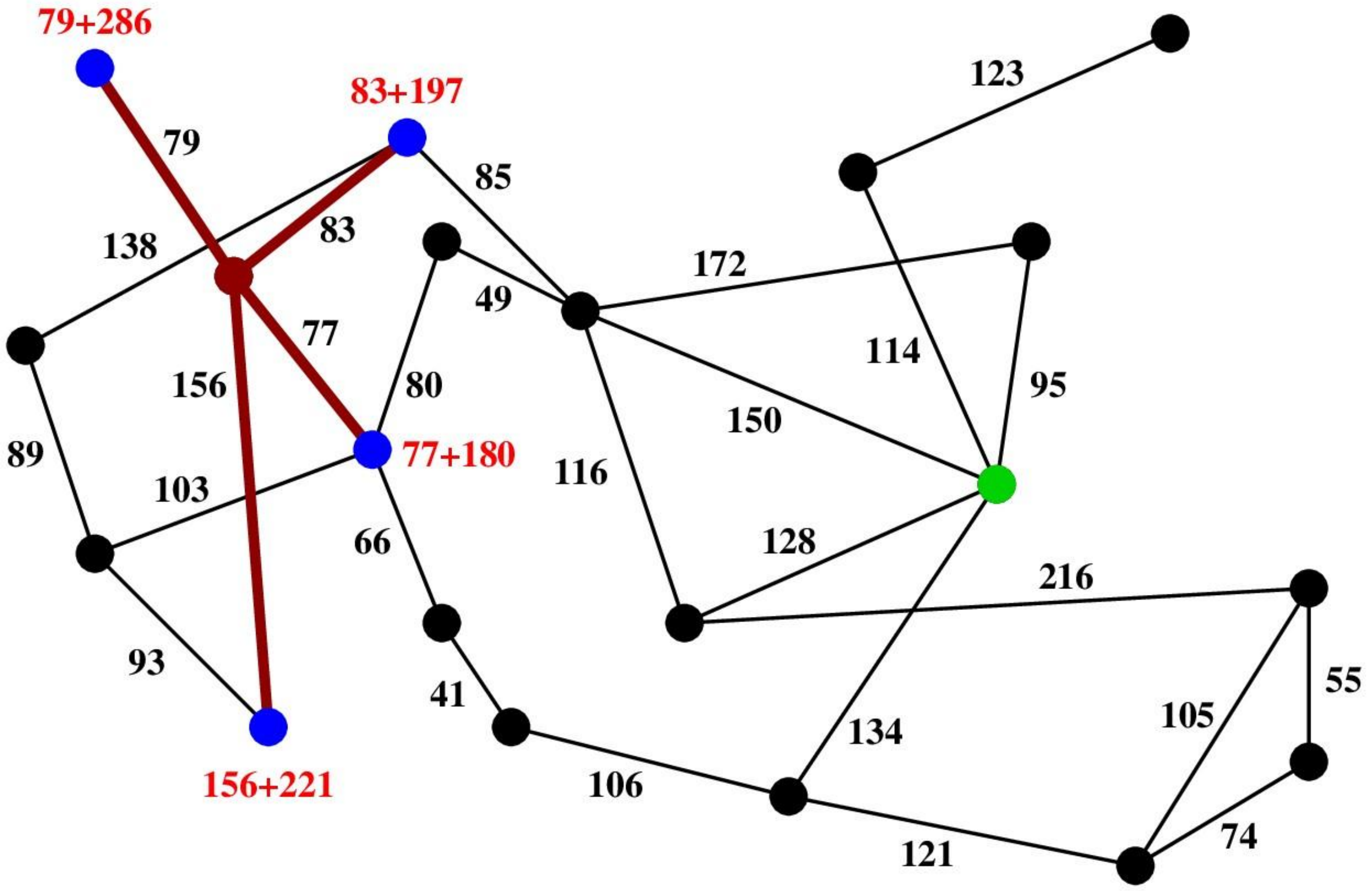
For suitable heuristics $h(n)$, A* search is cost-optimal.

Example of A* search

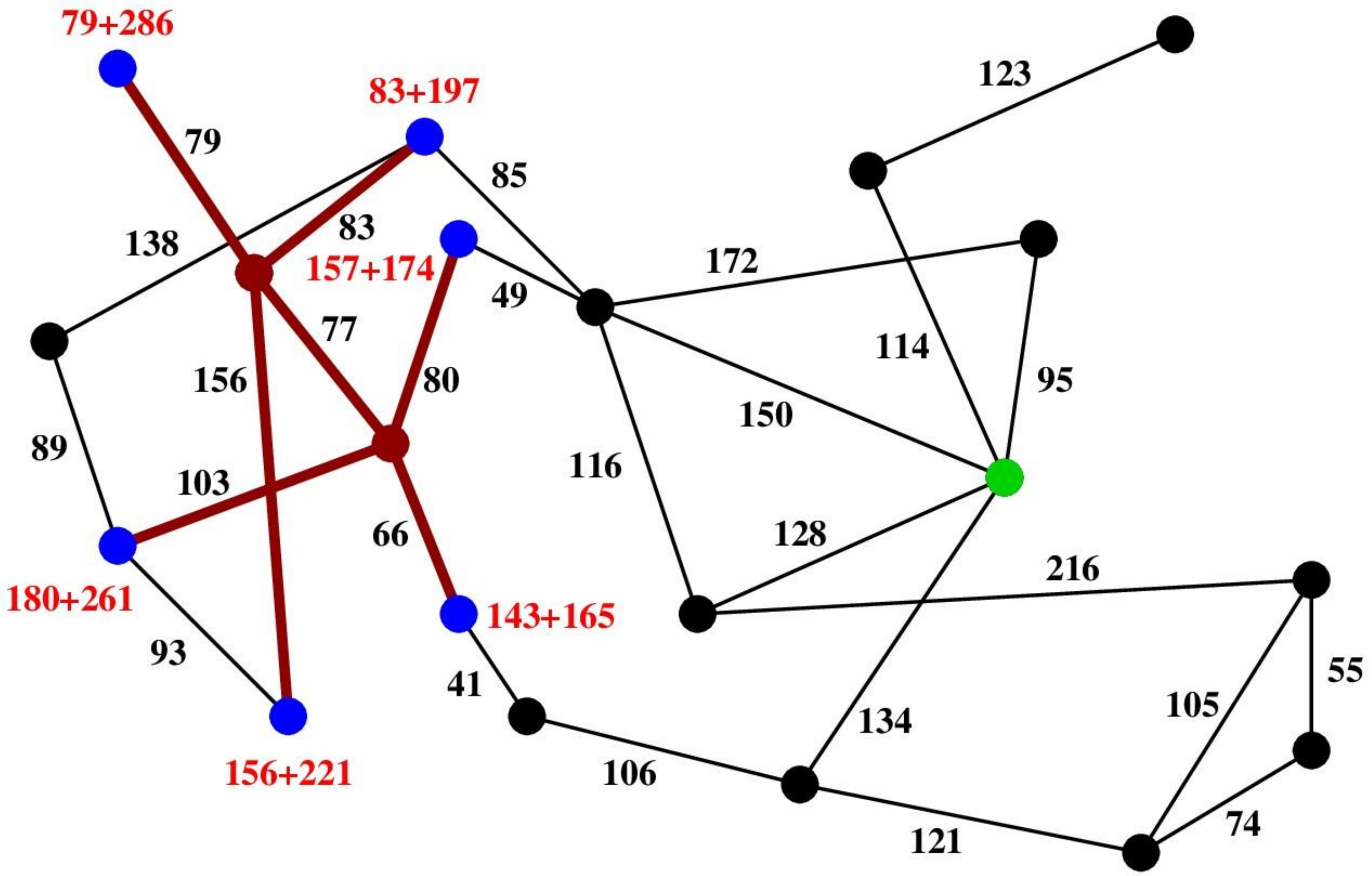


h(*n*) is the straight line distance between *n* and the goal node.

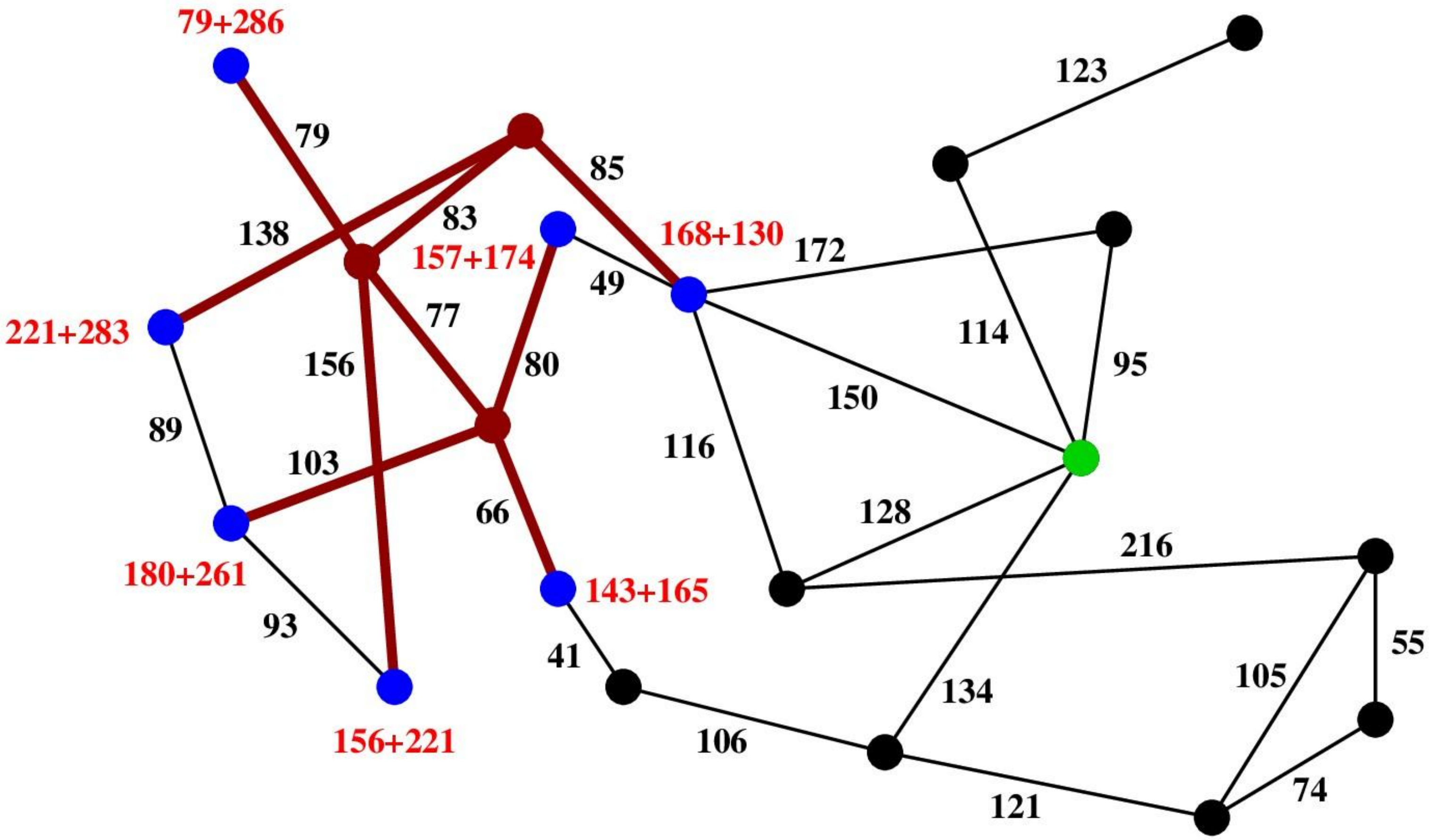
Example of A* search



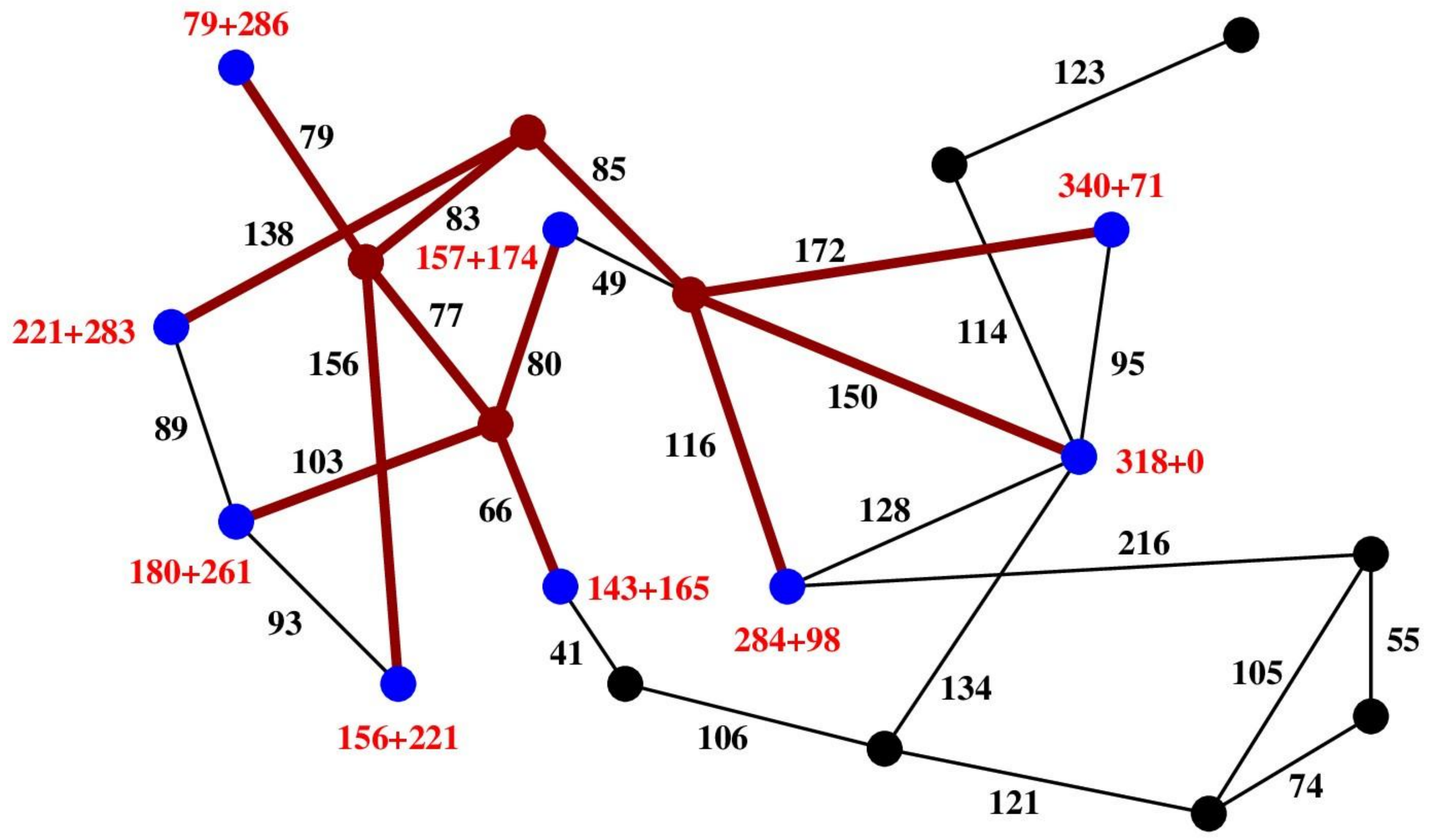
Example of A* search



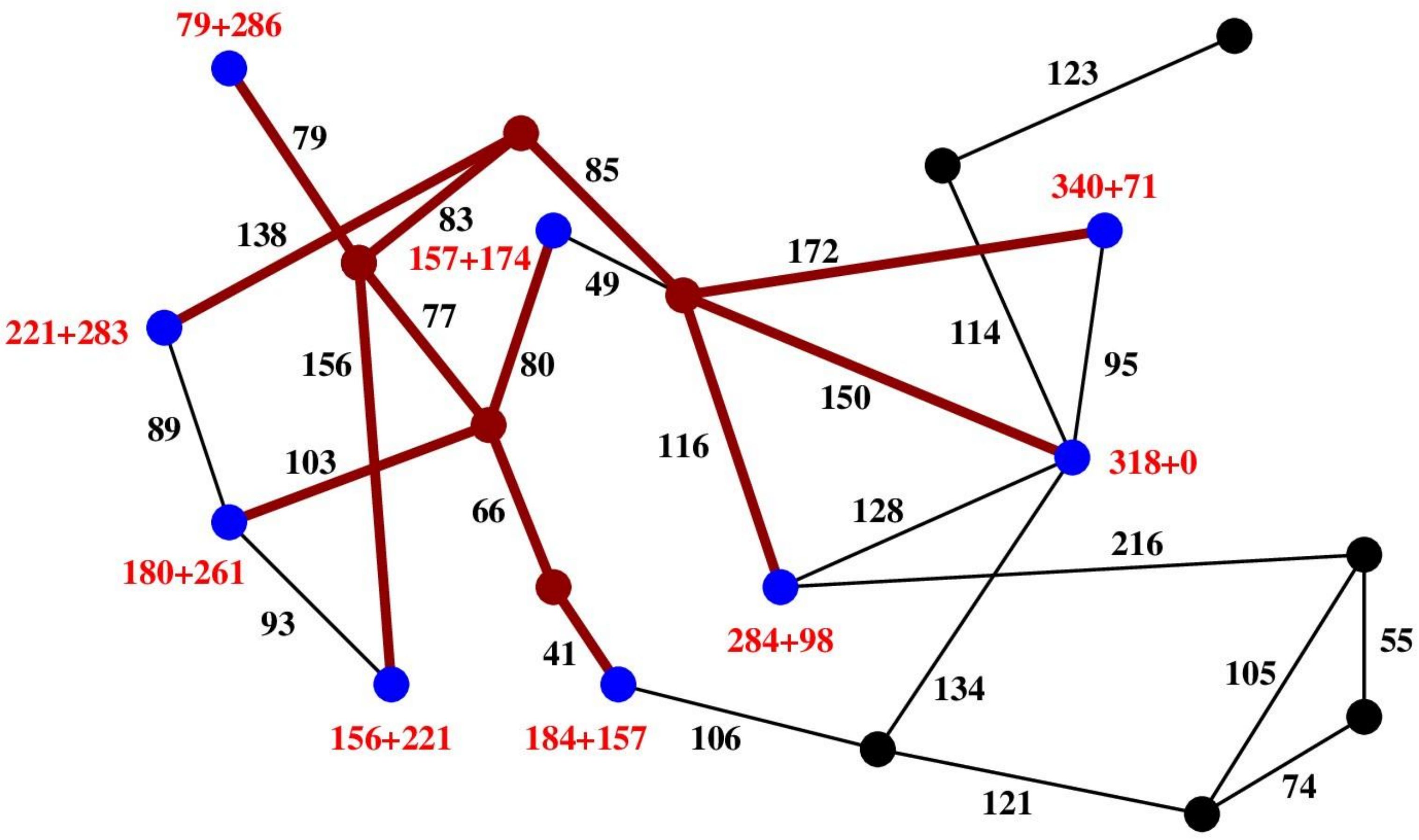
Example of A* search



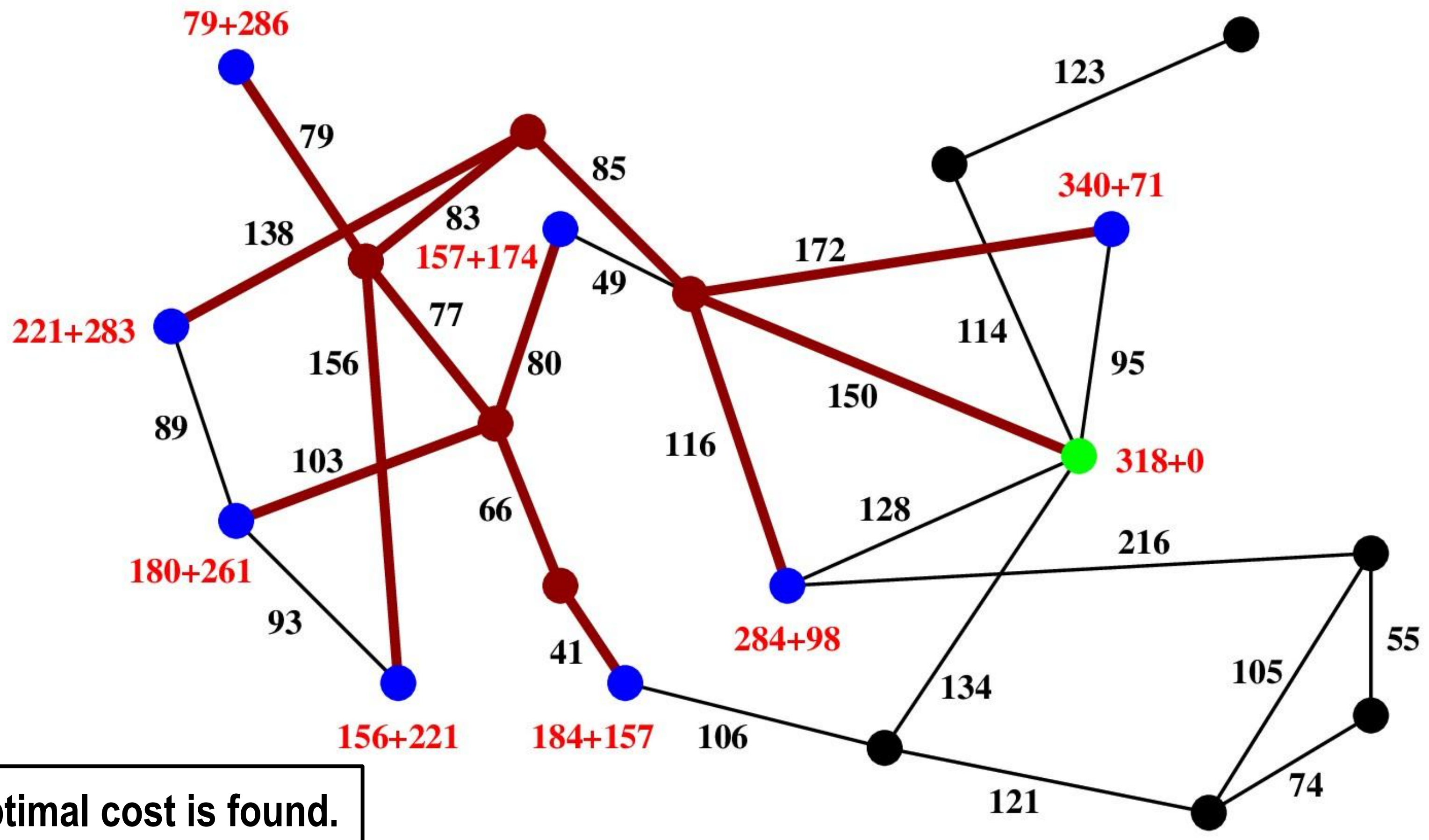
Example of A* search



Example of A* search



Example of A* search



The optimal cost is found.

Admissible / Optimistic heuristic

$g^*(n)$ = the cost of the optimal path from root to n.

$h^*(n)$ = cost of the optimal path from n to goal.

The heuristic is called **admissible** if $h(n) \leq h^*(n)$ for all n.

An admissible heuristic never overestimates the remaining cost.

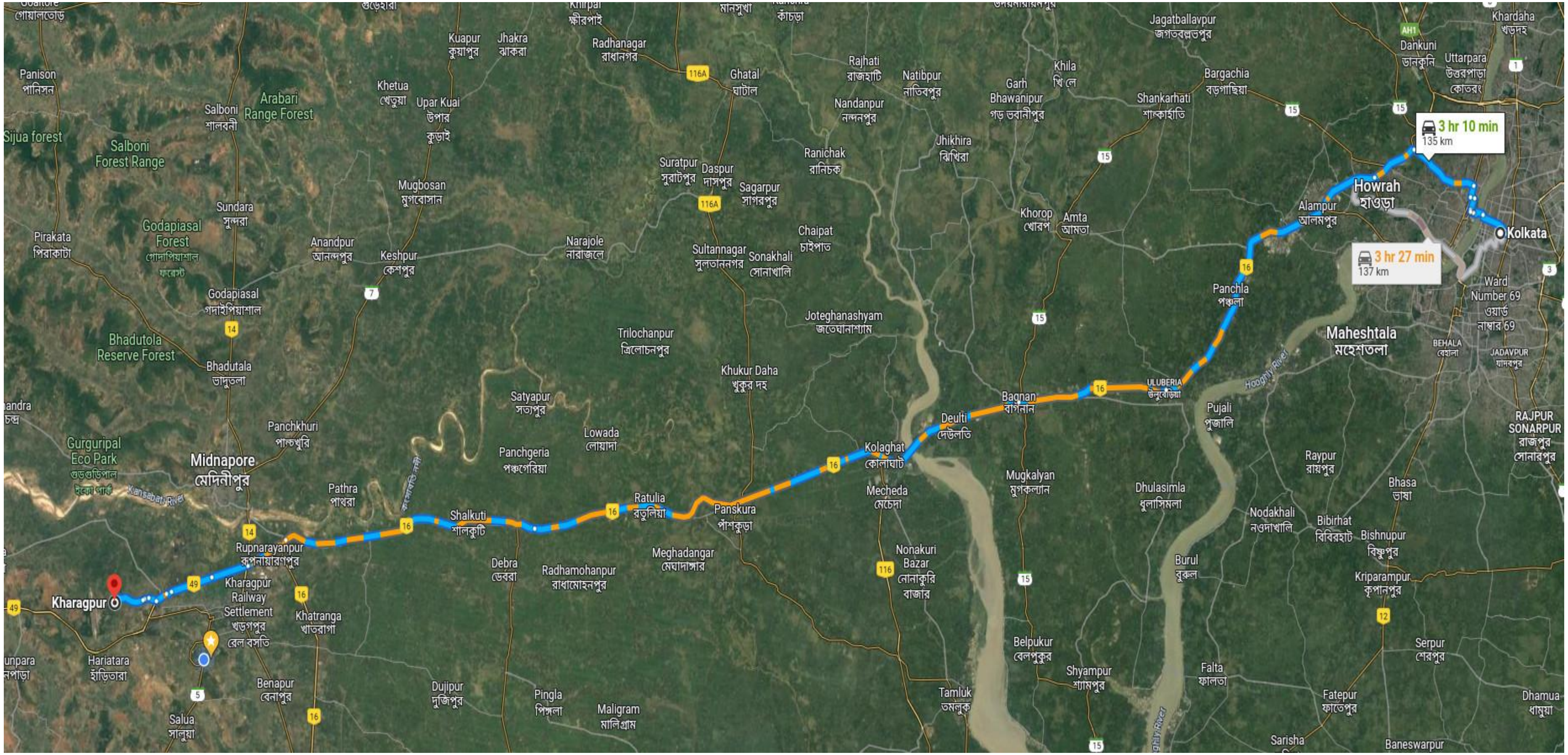
Theorem: If $h(n)$ is admissible, then A* search finds a cost-optimal solution (in finite state spaces).

Example: The straight line distance between pairs of cities

- If two cities are directly connected, the road rarely follows the straight line path.
- If there are detours through other cities, the road length may be higher.

Corollary: Our A* example found the cheapest path not by luck, but by design.

Road connecting Kharagpur with Kolkata [Google Maps]



Proof of cost-optimality of A* search under an admissible heuristic

- Let C^* be the optimal cost from the start node S to a goal node G.
- Let P^* be an optimal $S \longrightarrow G$ path (so cost(P^*) = C^*).
- Suppose that A* search gives an $S \longrightarrow G$ path P of cost $C > C^*$.
- Not all non-G nodes on P^* are expanded. Otherwise the optimal $S \longrightarrow G$ path P^* (or another optimal path of the same cost C^* must have been discovered, which contradicts $C > C^*$).
- Let n be the *first* open node on P^* (that is n is generated but not expanded).
- Since n and G are both open when G is extracted from the frontier, we have $f(n) \geqslant C > C^*$.
- $f(n) = g(n) + h(n)$ (by definition of f).
- $g(n) = g^*(n)$ (For each node m on P^*, the $S \longrightarrow m$ subpath of P^* is a shortest $S \longrightarrow m$ path (why?). Also, since n is the first open node on P^*, all nodes on P^* preceding n are closed (no such node is open or not generated). Therefore, a shortest $S \longrightarrow n$ path is already discovered.)
- $h(n) \leqslant h^*(n)$ (because h is admissible).
- The cost of the $n \longrightarrow G$ subpath of P^* is $h^*(n)$ (why?).
- $C^* < C \leqslant f(n) = g(n) + h(n) \leqslant g^*(n) + h^*(n) = \text{cost}_{P^*}(S \longrightarrow n) + \text{cost}_{P^*}(n \longrightarrow G) = \text{cost}(P^*) = C^*$, a contradiction.

Exercise: Does this proof also establish the following assertion?

Whenever a node v is expanded by A, a shortest $S \longrightarrow v$ path is already discovered.*

Consistent heuristics

Triangle inequality:

For every transaction $n \longrightarrow n'$, we have $h(n) \leqslant \text{cost}(n,n') + h(n')$.

Consistent $\Longrightarrow$ Admissible

Assume that $h(G) = 0$ for all goal nodes G.

Take any non-goal node n. Let $P = (n, n_1, n_2, \ldots, n_k = G)$ be an optimal $n \longrightarrow G$ path for some goal node G. By consistency, we have $h(n) \leqslant c(n,n_1) + h(n_1) \leqslant c(n,n_1) + c(n_1,n_2) + h(n_2) \leqslant \ldots \leqslant c(n,n_1) + c(n_1,n_2) + \ldots + c(n_{k-1},n_k) = h^*(n)$.

The converse is not true

Consider the graph $S \longrightarrow A \longrightarrow B \longrightarrow G$ with all transition costs equal to 1.
Take $h(S) = 3$, $h(A) = h(B) = 1$ and $h(G) = 0$.

Consistent heuristics

Theorem: When a node n is expanded, A* search (under a consistent heuristic) has found a shortest $S \longrightarrow n$ path.

Let P be the $S \longrightarrow n$ path discovered by the algorithm. Let P^* be a shortest $S \longrightarrow n$ path. Assume that $P \neq P^*$. Let $n' \neq n$ be the first node on P^*, that is not expanded yet (but generated). We have:

$h(n') \leqslant h(n) + c_{P^*}(n' \longrightarrow n)$ [by consistency of h]
$f(n) \leqslant f(n')$ [because n is expanded before n']
For every node m on P^*, the subpath $S \longrightarrow m$ is an optimal path.

But then

$$g(n) = f(n) - h(n) \leqslant f(n') - h(n) = g(n') + h(n') - h(n) \leqslant g(n') + c_{P^*}(n' \longrightarrow n)$$
$$= g^*(n') + c_{P^*}(n' \longrightarrow n) = g^*(n).$$

That is, $g(n) = g^*(n)$, that is, P is a shortest $S \longrightarrow n$ path too.

Corollary: A* search under a consistent heuristic never needs to reopen a closed node.

Weighted A* Search

A* search may generate a lot of nodes.

Taking $f(n) = g(n) + W \times h(n)$ for some $W > 1$ may convert an admissible heuristic non-admissible.

But an appropriate weight W may supply more realistic estimates (than with $W = 1$).

Example: Detour index on road networks is about 1.3. So we can take $W = 1.3$.

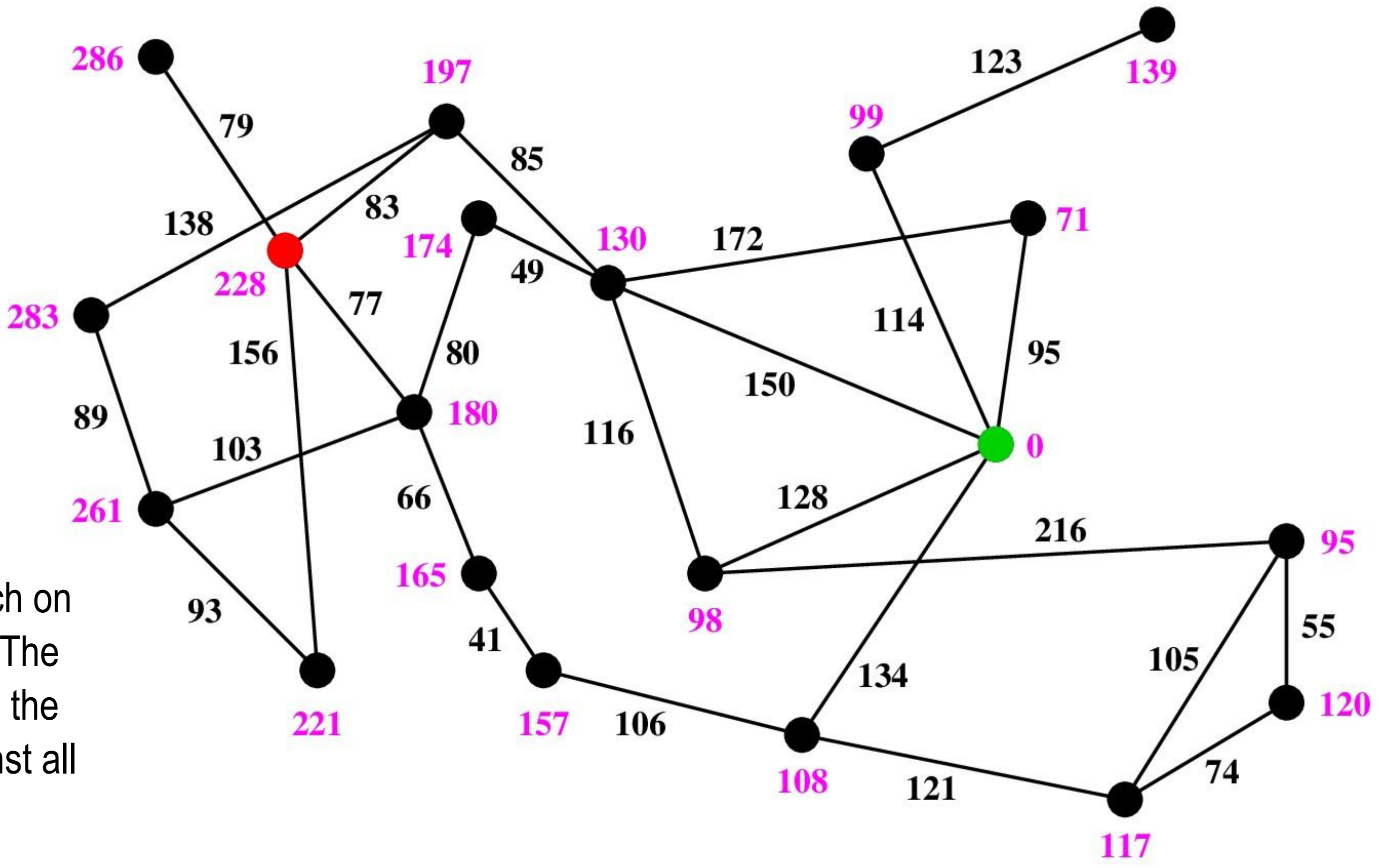
But this means that $h(n)$ may sometimes overestimate $h^*(n)$.

So an optimum solution is no longer guaranteed.

We should be happy with a **suboptimal** (also called **satisficing**) solution.

But there may be a significant saving in the number of nodes generated.

Weighted A* Search: Exercise



Run the Weighted A* search on our example with *W* = 1.3. The straight line distances from the goal node are shown against all the nodes.

Iterative Deepening A* (IDA*) Search

Too many nodes are generated and **remembered** by BFS, Uniform-Cost Search, and A* search.

The search may run out of memory.

DFS generates much fewer nodes by not remembering reached states (except along the search path), but may end up in infinite paths.

IDS restricts the maximum depth of DFS and eliminates infinite paths.

But IDS generates nodes multiple times (across different iterations).

IDS is still a memory-efficient solution to the uninformed search problem.

IDA* is the memory-efficient analog for A*.

The *f*() values are used to restrict the depth of DFS and avoid infinite paths.

IDA* algorithm

The Algorithm

- Let F be a limit on $f()$ values.
- Run a DFS from the start state. But never expand any (generated) node n with $f(n) > F$.
- If the goal is reached, we are done. Otherwise increase F, and restart the search.

How to choose *F* in different iterations?

- **First iteration:** Take $F = f(S)$, where S is the start node.
- **Later iterations:** All nodes n reachable from S and with $f(n) \leqslant F$ are expanded. Take

$$\min \left(f(n) \,\middle|\, n \text{ is generated and } f(n) > F \right)$$

Performance of IDA*

Because reached nodes are not remembered, a node *n* may be generated multiple times.

If any of these generations gives $f(n) \leqslant F$, then *n* is expanded.

Multiple generation of the same nodes does not affect the minimum computation to determine *F* for the next iteration.

Each iteration expands all nodes *n* with $f(n) \leqslant F$.

But the number of iterations may be large.

It may be that in each iteration, only one additional node is expanded.
(Example: Run IDA* on our path-finding example.)

So the number of iterations may be as large as the number of nodes in the search space.

Recursive Best-First Search (RBFS)

IDA* (also DFS, IDS) maintains only a single path of the internal nodes starting from the root along with the siblings of all the nodes on that path.

This is how IDA* achieves memory efficiency.

However, IDA* suffers from many node re-generations leading to inefficiency.

An improved algorithm is RBFS.

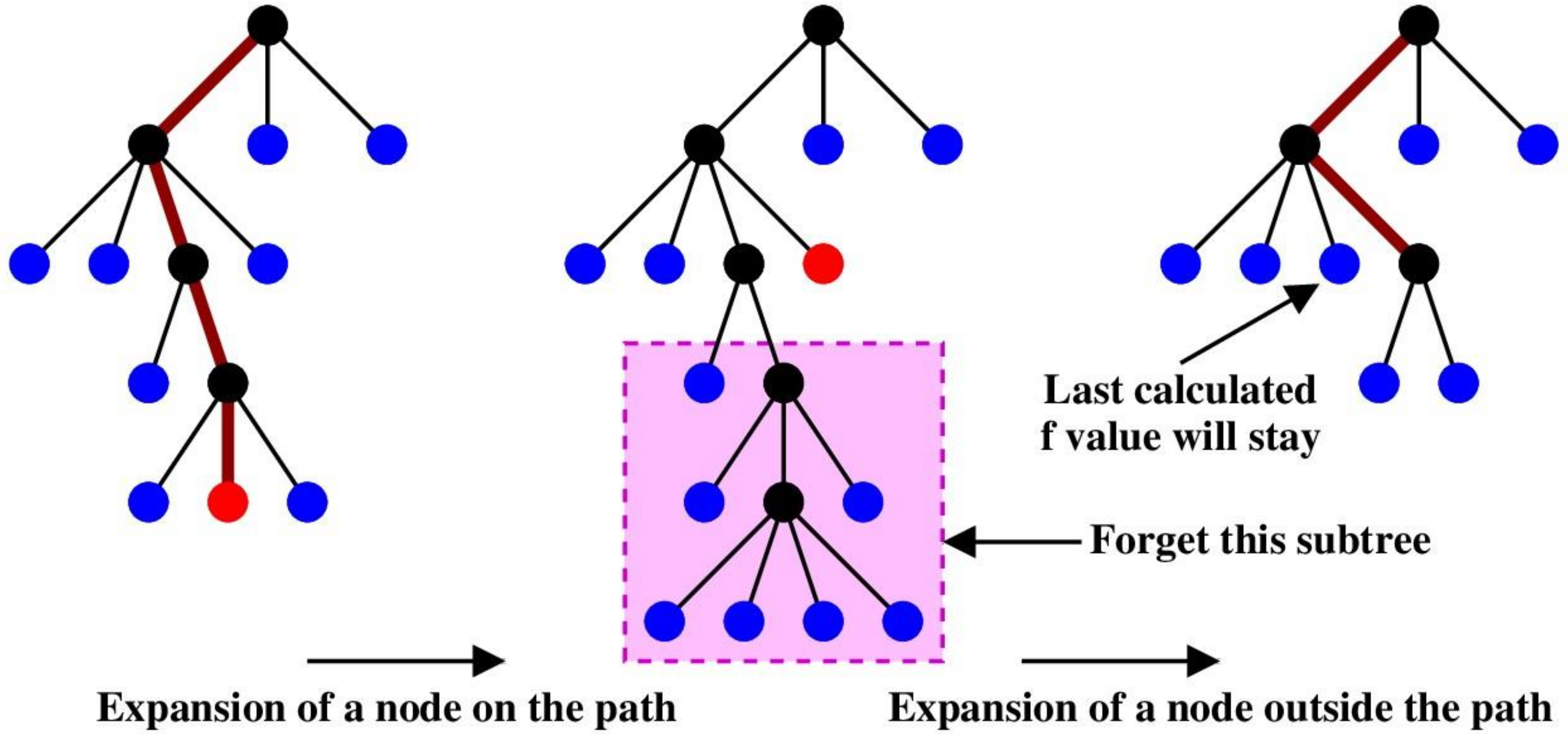
RBFS too maintains a single path of internal nodes plus the siblings of the nodes on that path.

The $f()$ values stored in a node n is now not necessarily equal to $g(n) + h(n)$.

For a **newly generated leaf node**, this is the case.

For an **internal node** n, $f(n)$ is the minimum of the $f()$ values of all its child nodes. This is a more realistic estimate obtained after a few levels of expansion.

RBFS: Expansion explained [Reference: AINS]


Last calculated f value will stay
Forget this subtree
Expansion of a node on the path
Expansion of a node outside the path

RBFS: Expansion algorithm [Reference: AINS]

Let n be the cheapest node on the frontier.

Expand n to get all the child nodes of n.

Compute $f(m) = g(m) + h(m)$ for all child nodes m of n.

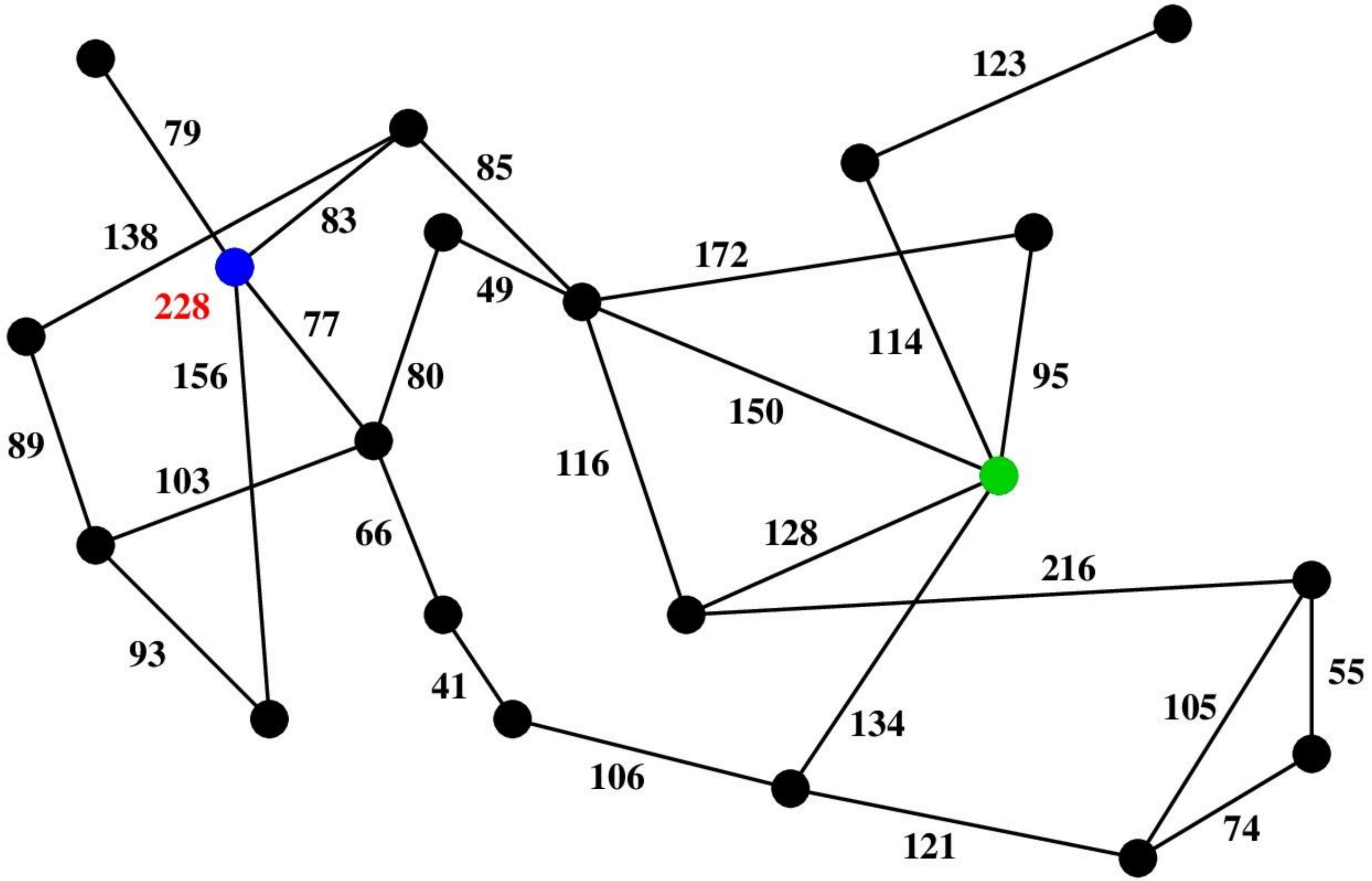
Update the $f()$ values on all internal nodes on the path maintained (n and all the ancestors of n).

If the new cheapest node is one of the child nodes of n, continue extending the path.

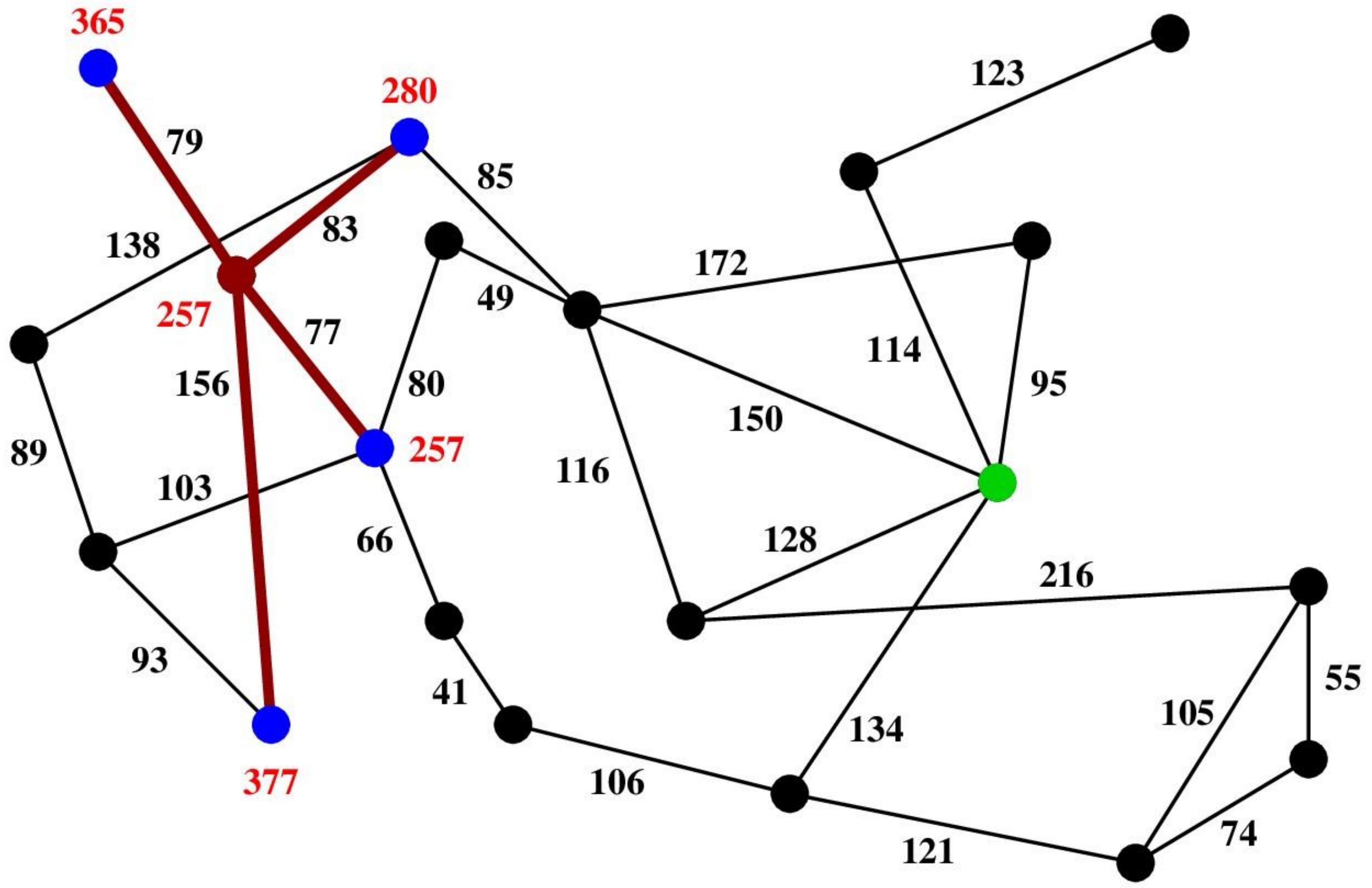
Otherwise:

> Let n' be the cheapest node on the frontier.
>
> Choose the lowest common ancestor u of n and n'.
>
> Let v be the node next to u on the $u \longrightarrow n$ path.
>
> Delete the subtree rooted at v. Keep v as an open node for future re-expansion.

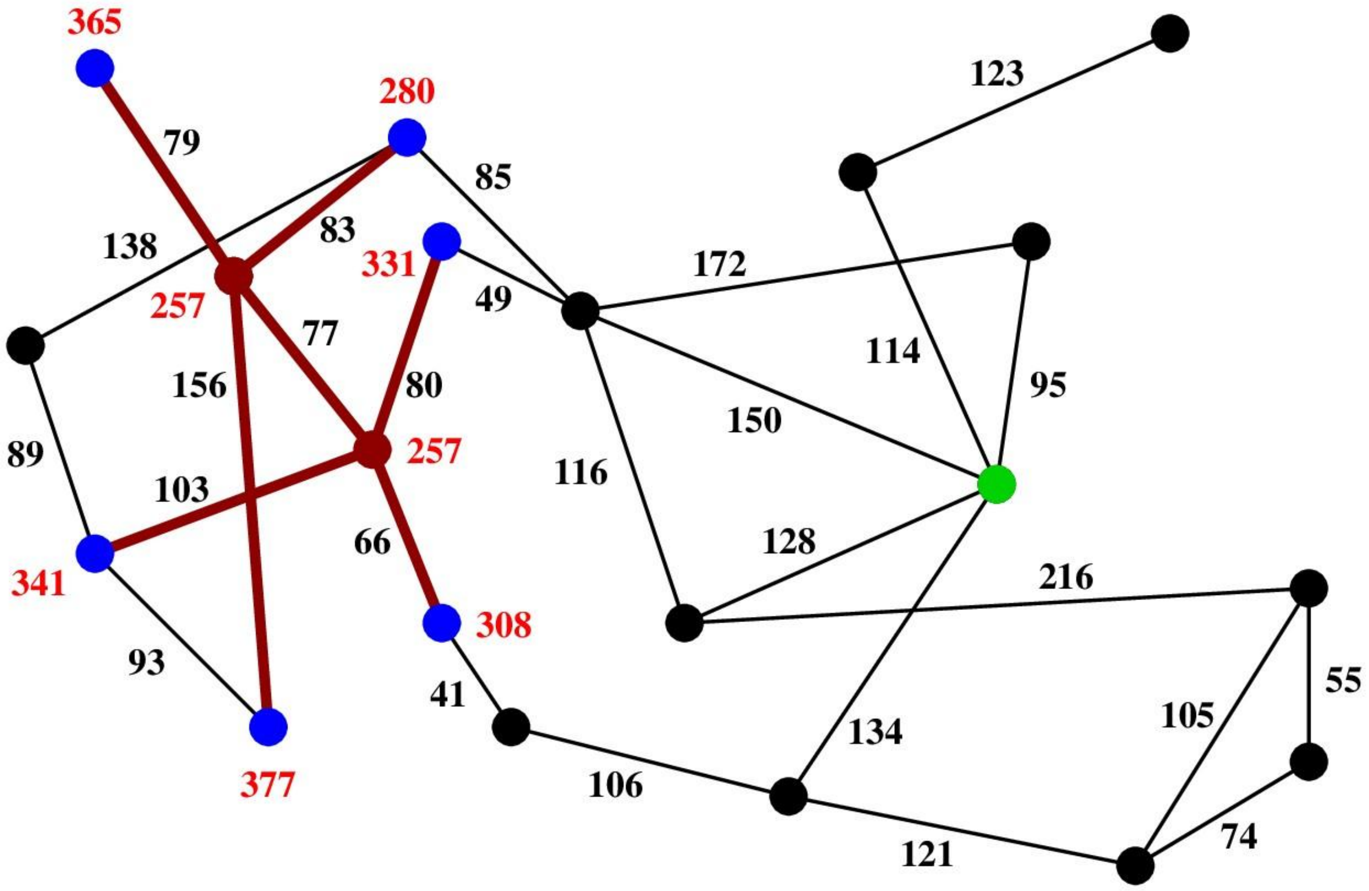
RBFS Example



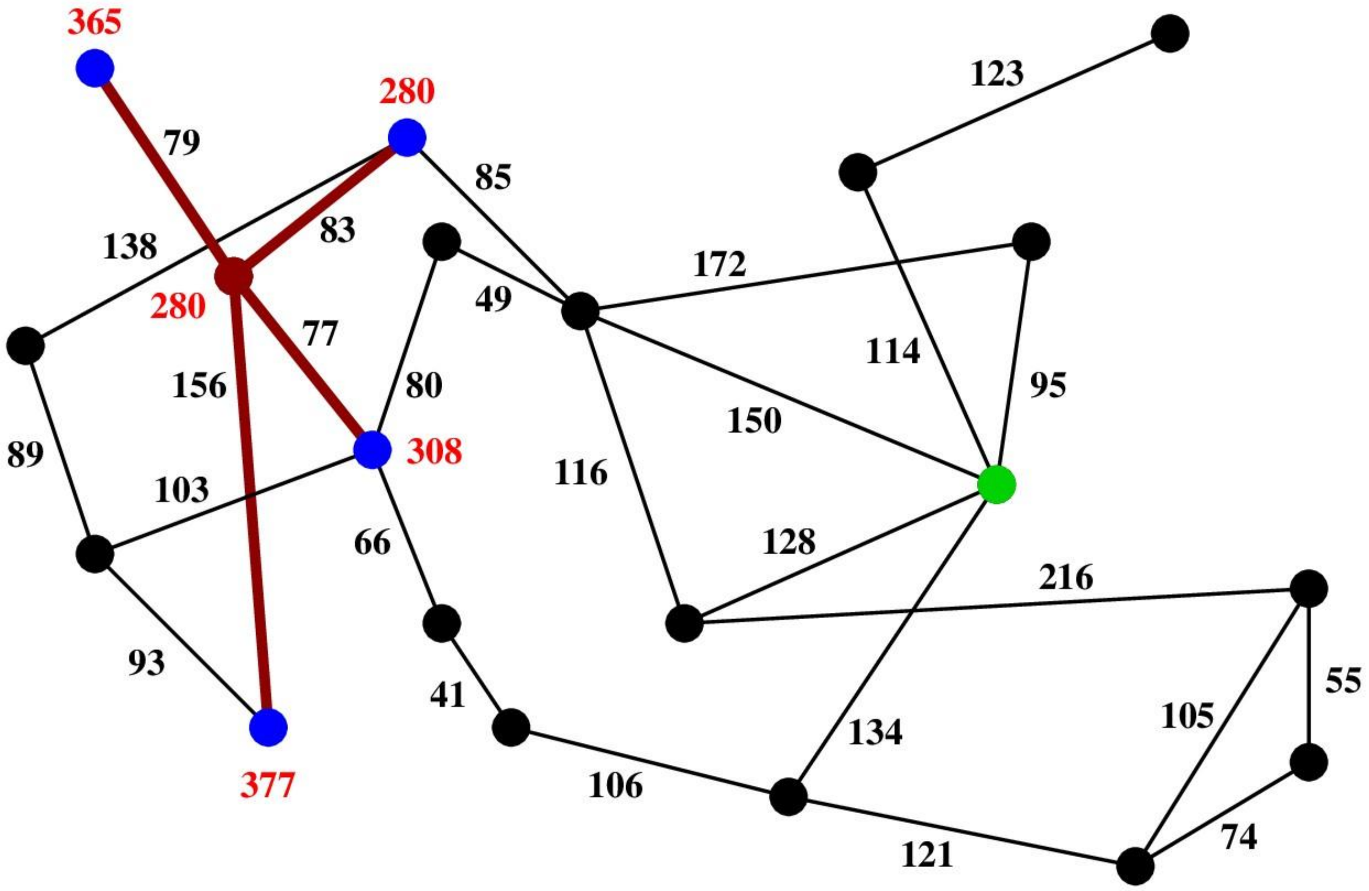
RBFS Example



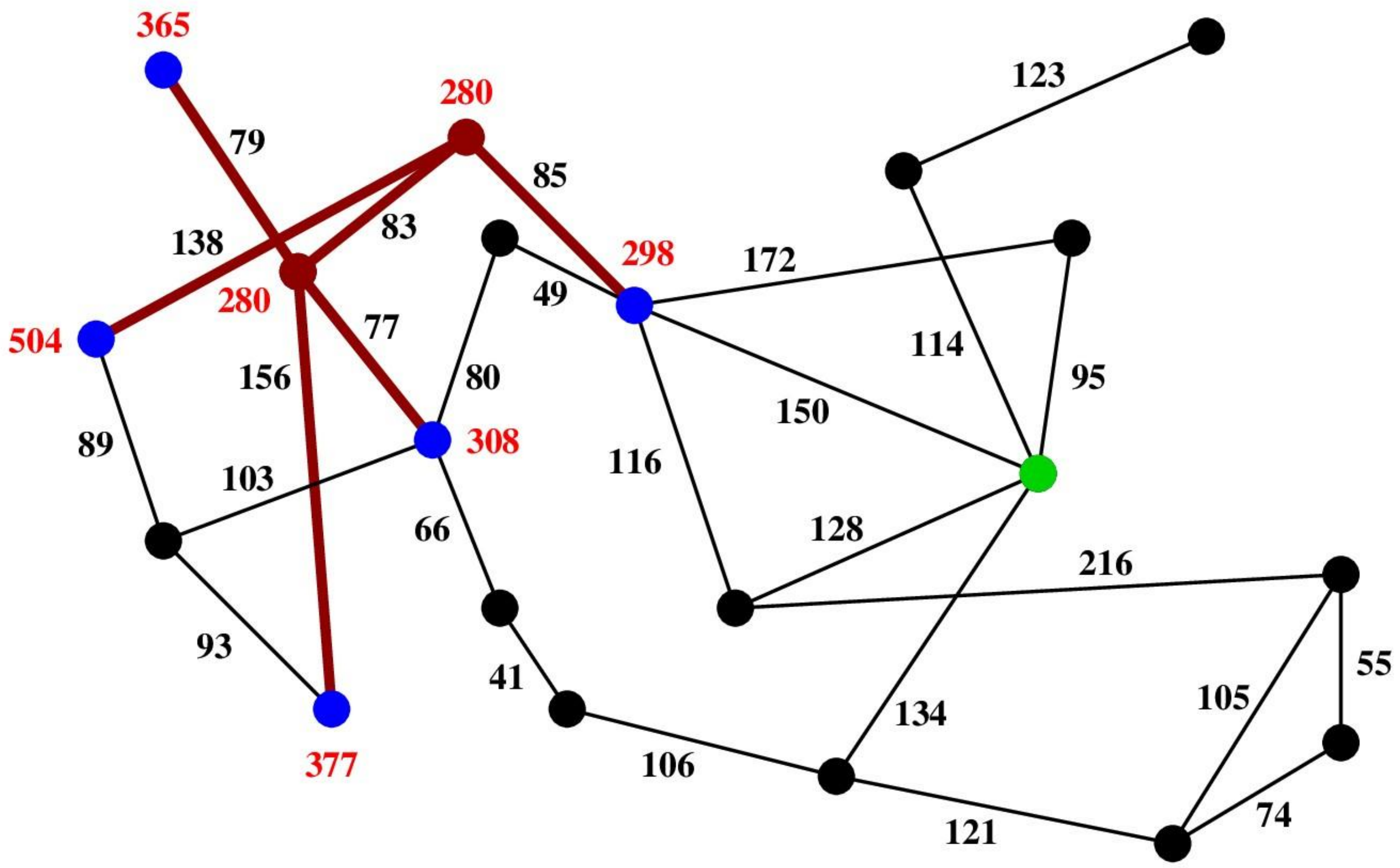
RBFS Example



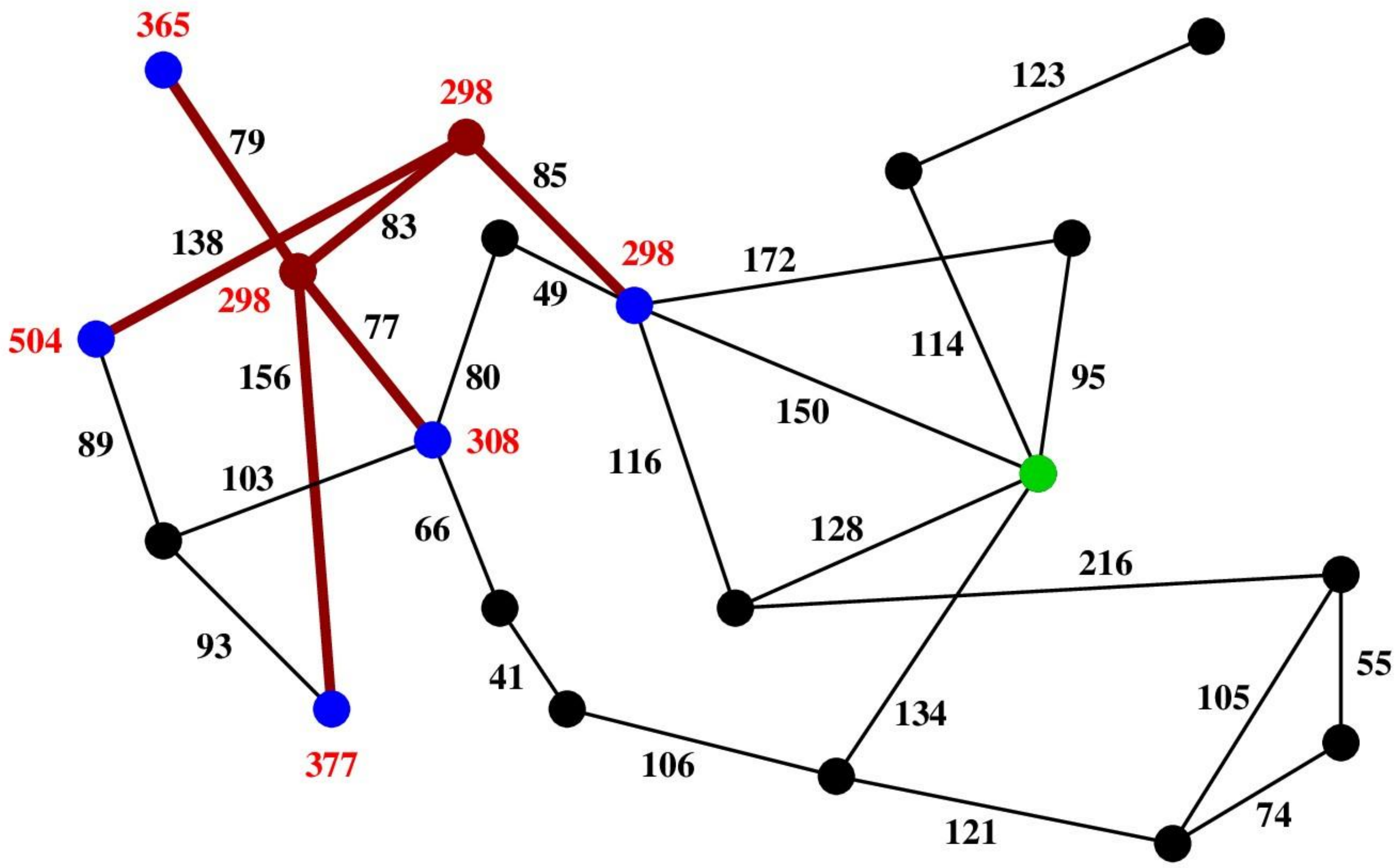
RBFS Example



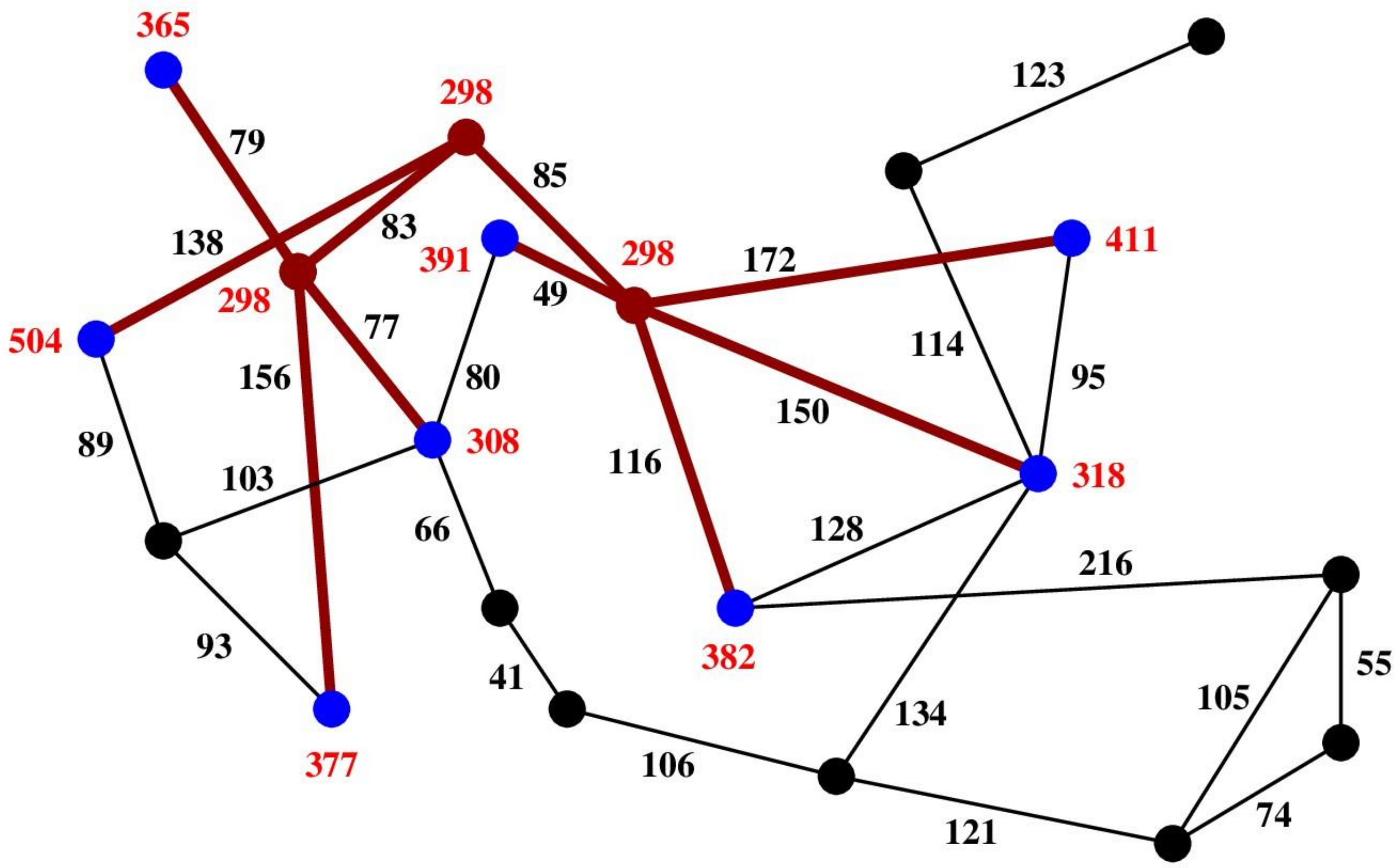
RBFS Example



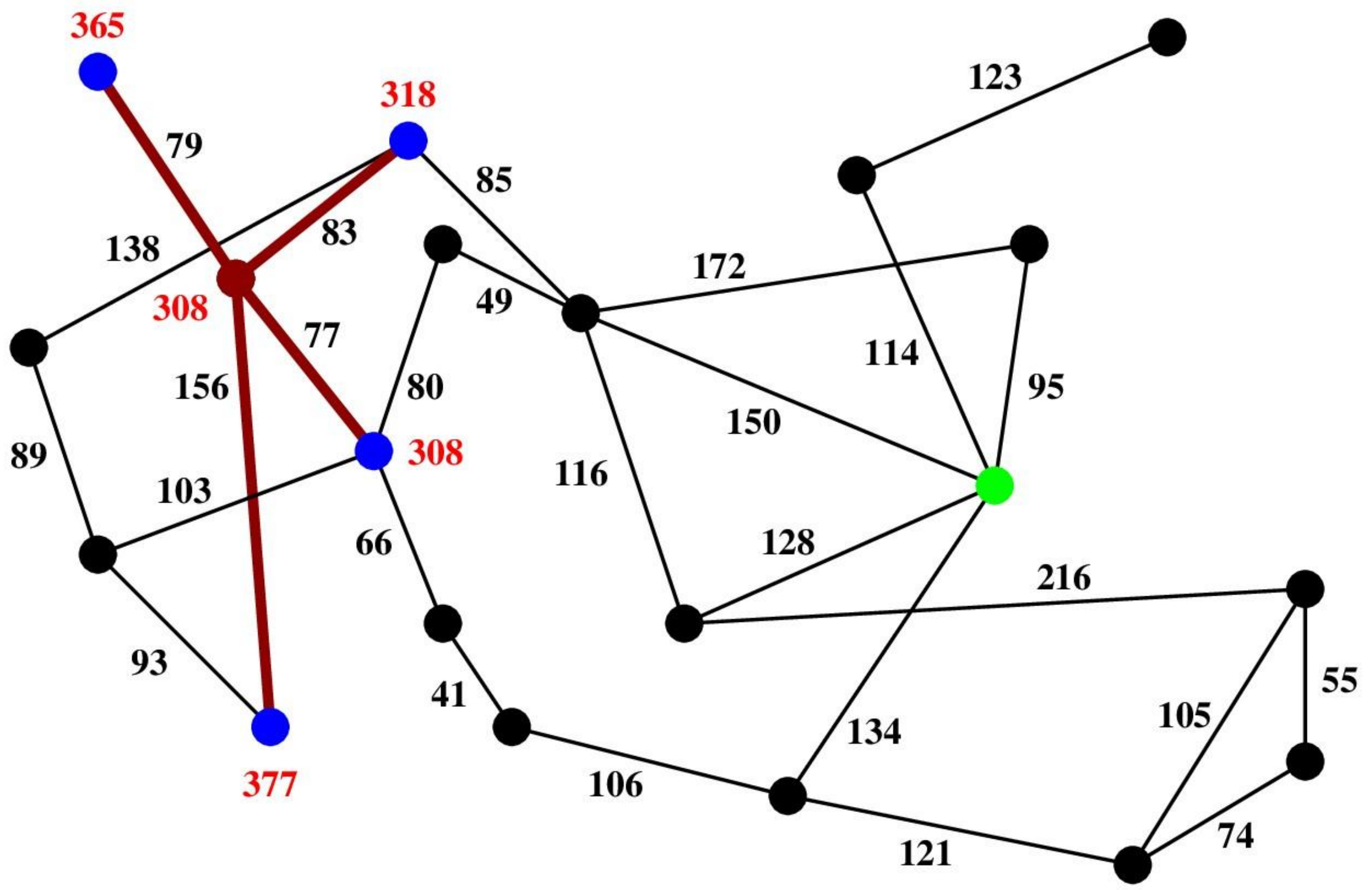
RBFS Example



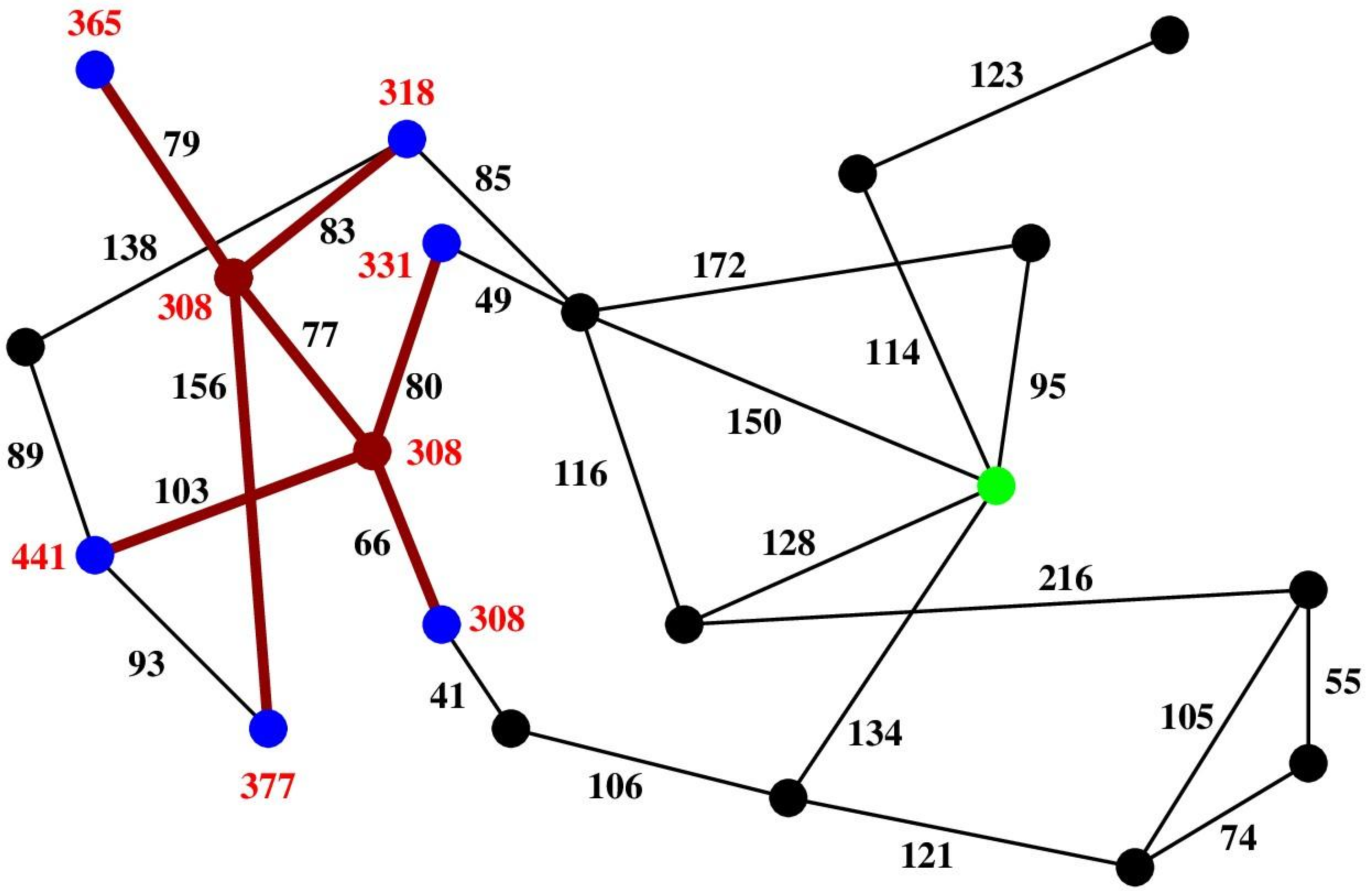
RBFS Example



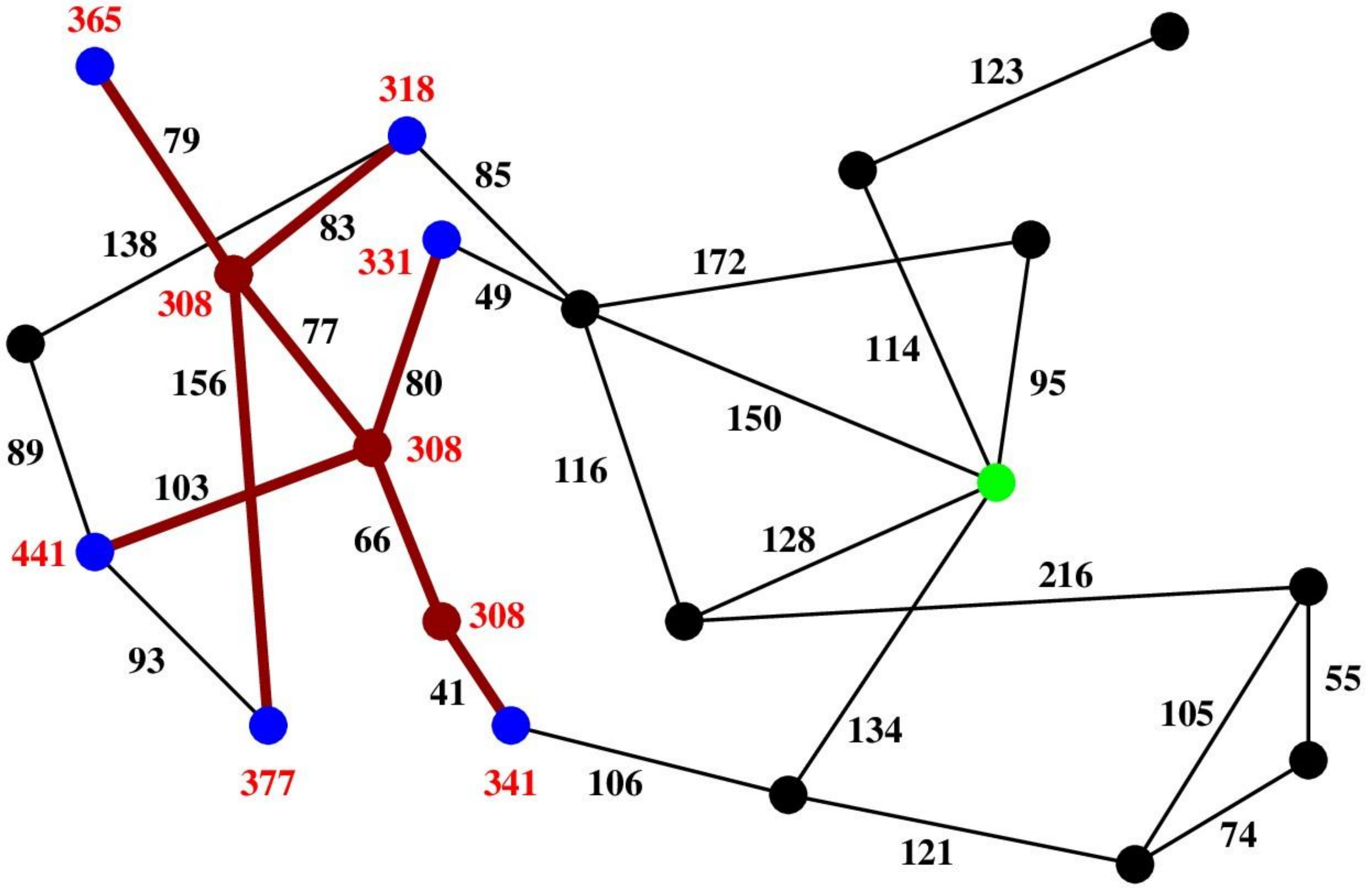
RBFS Example



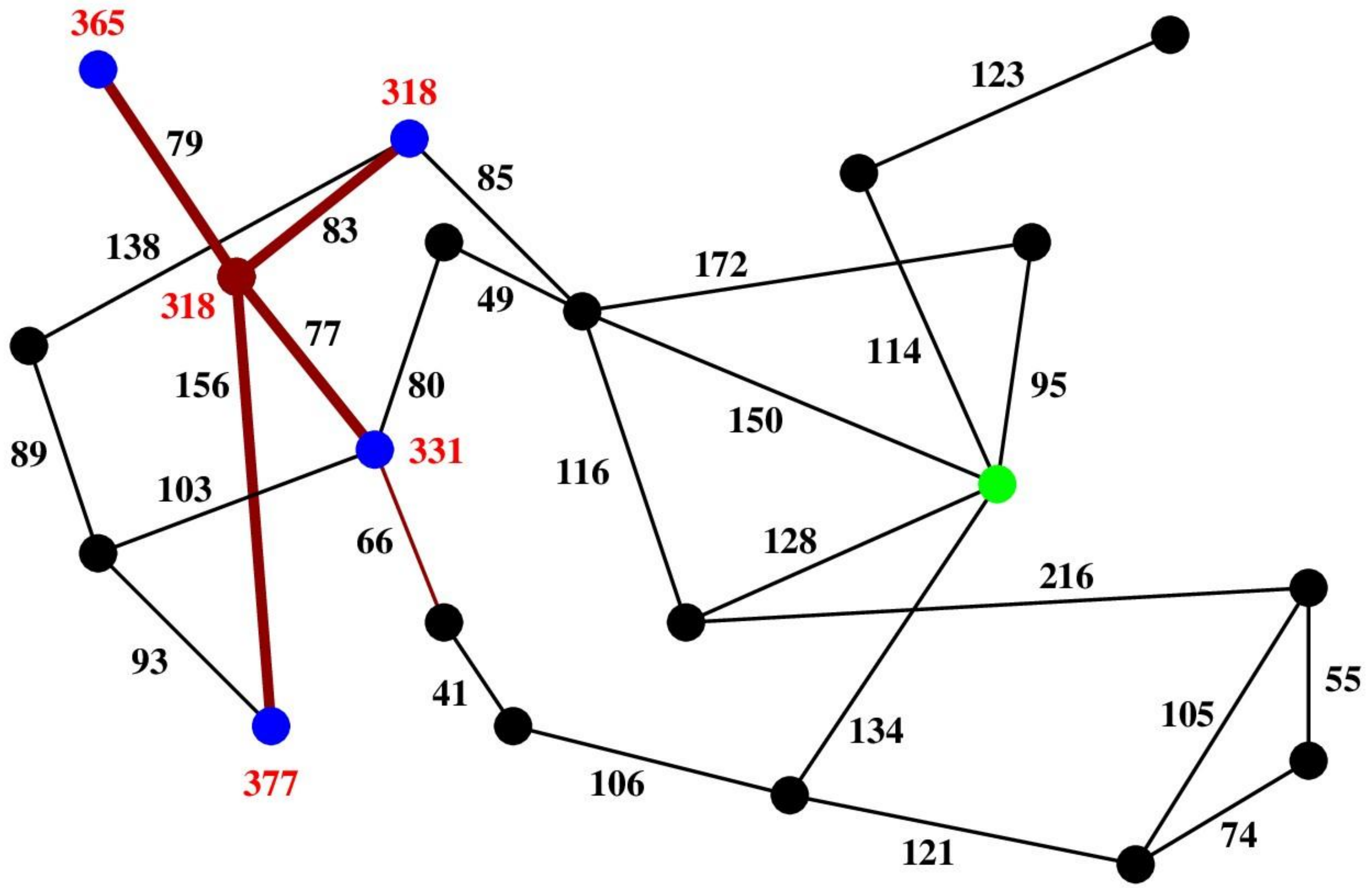
RBFS Example



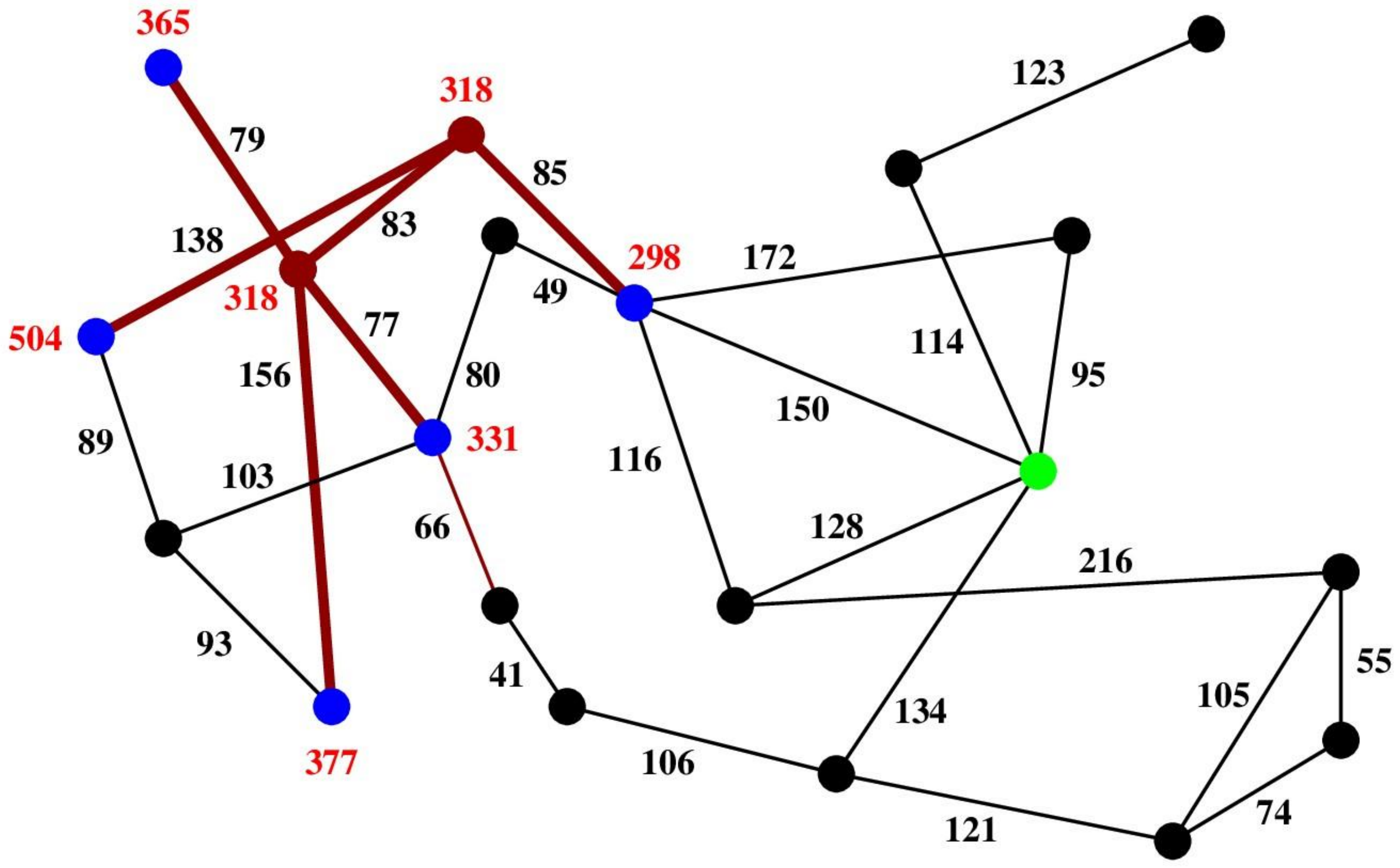
RBFS Example



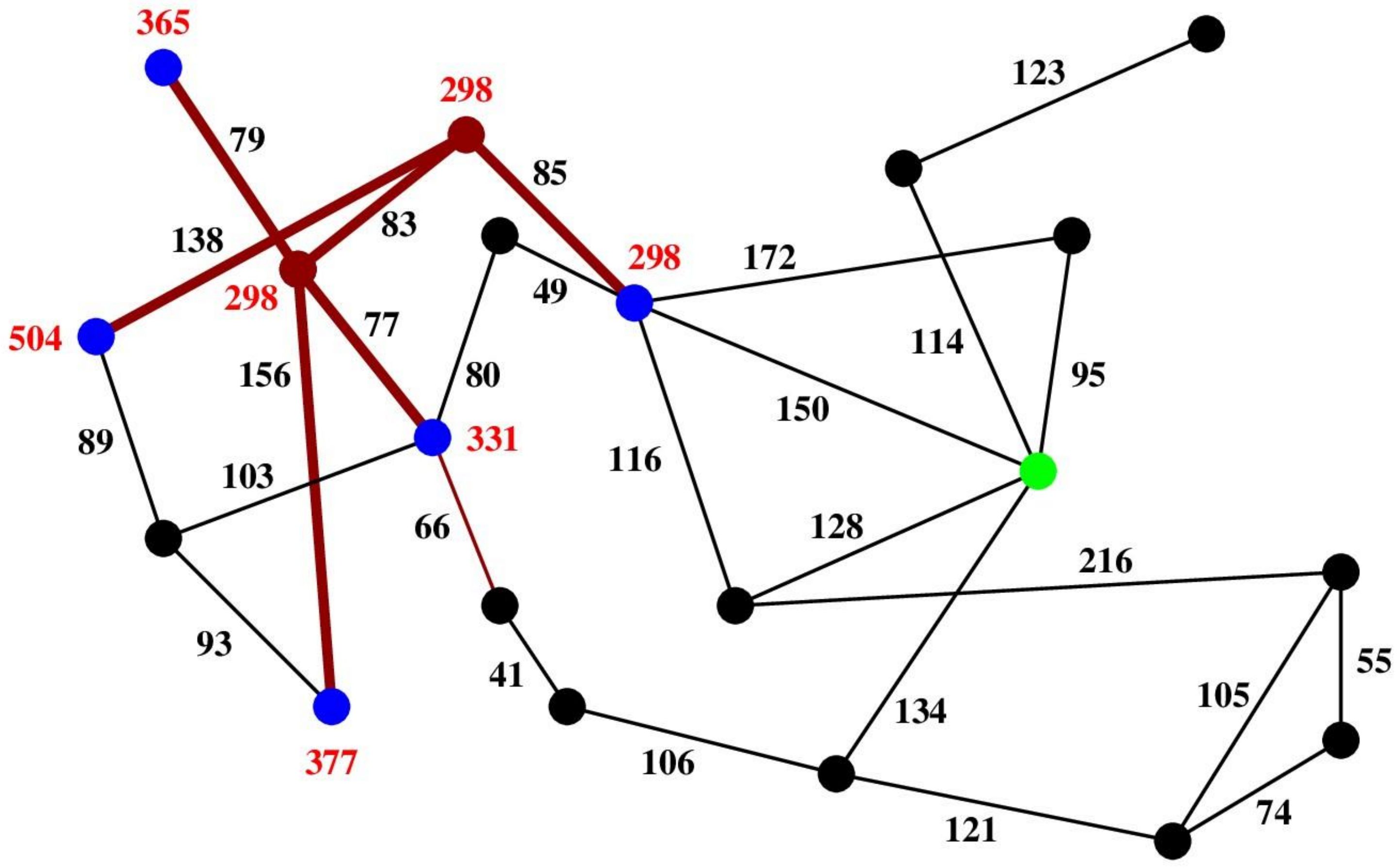
RBFS Example



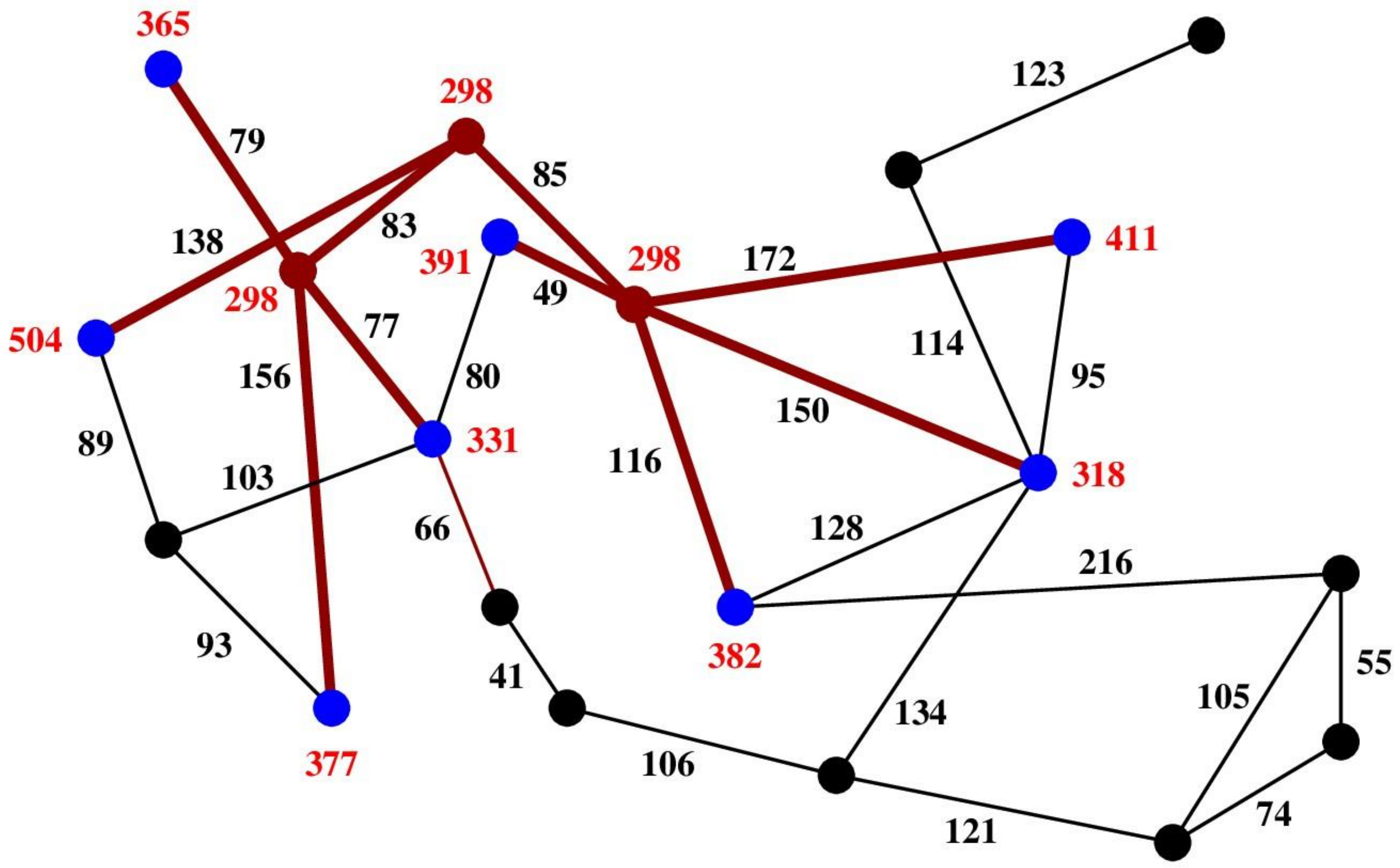
RBFS Example



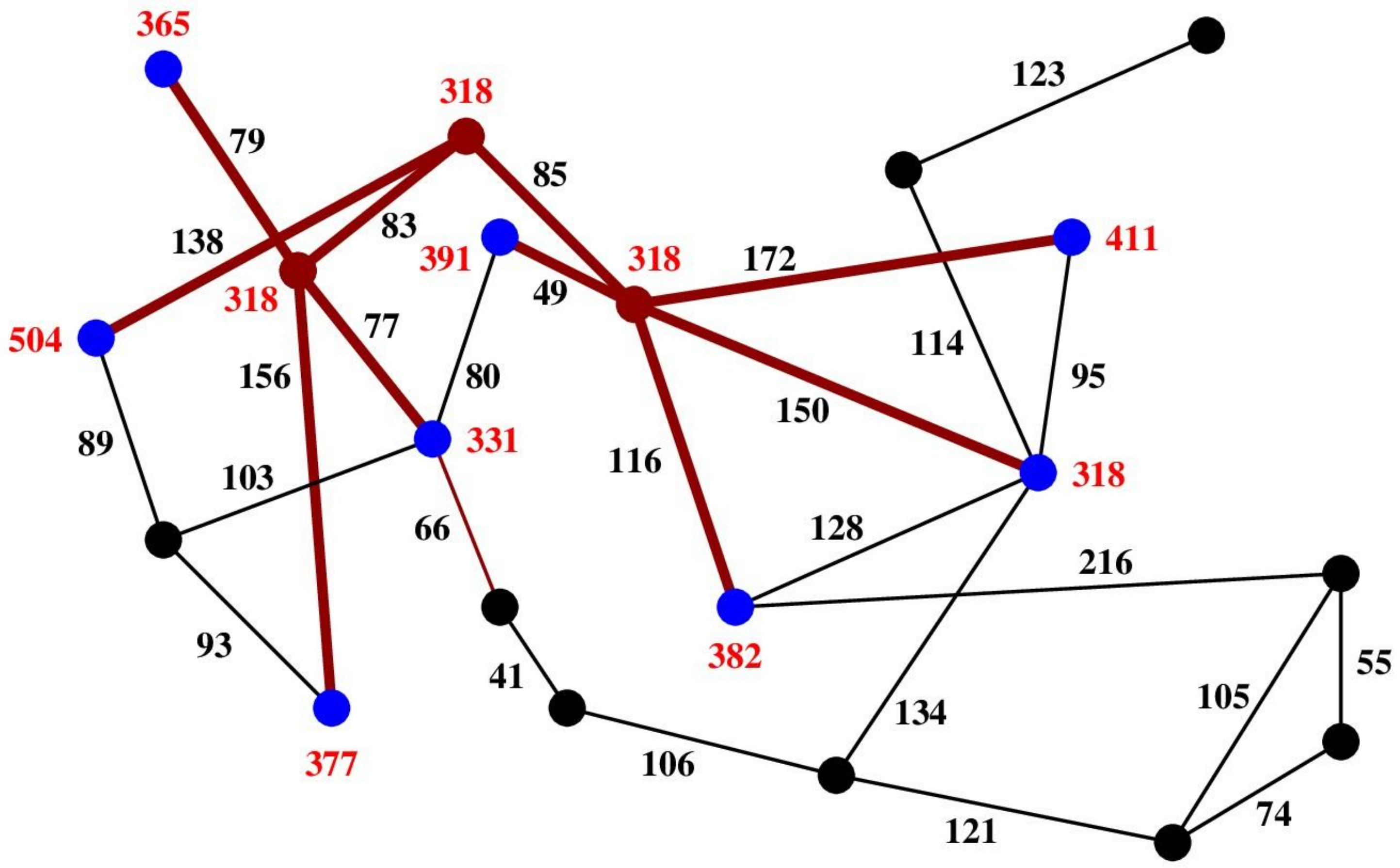
RBFS Example



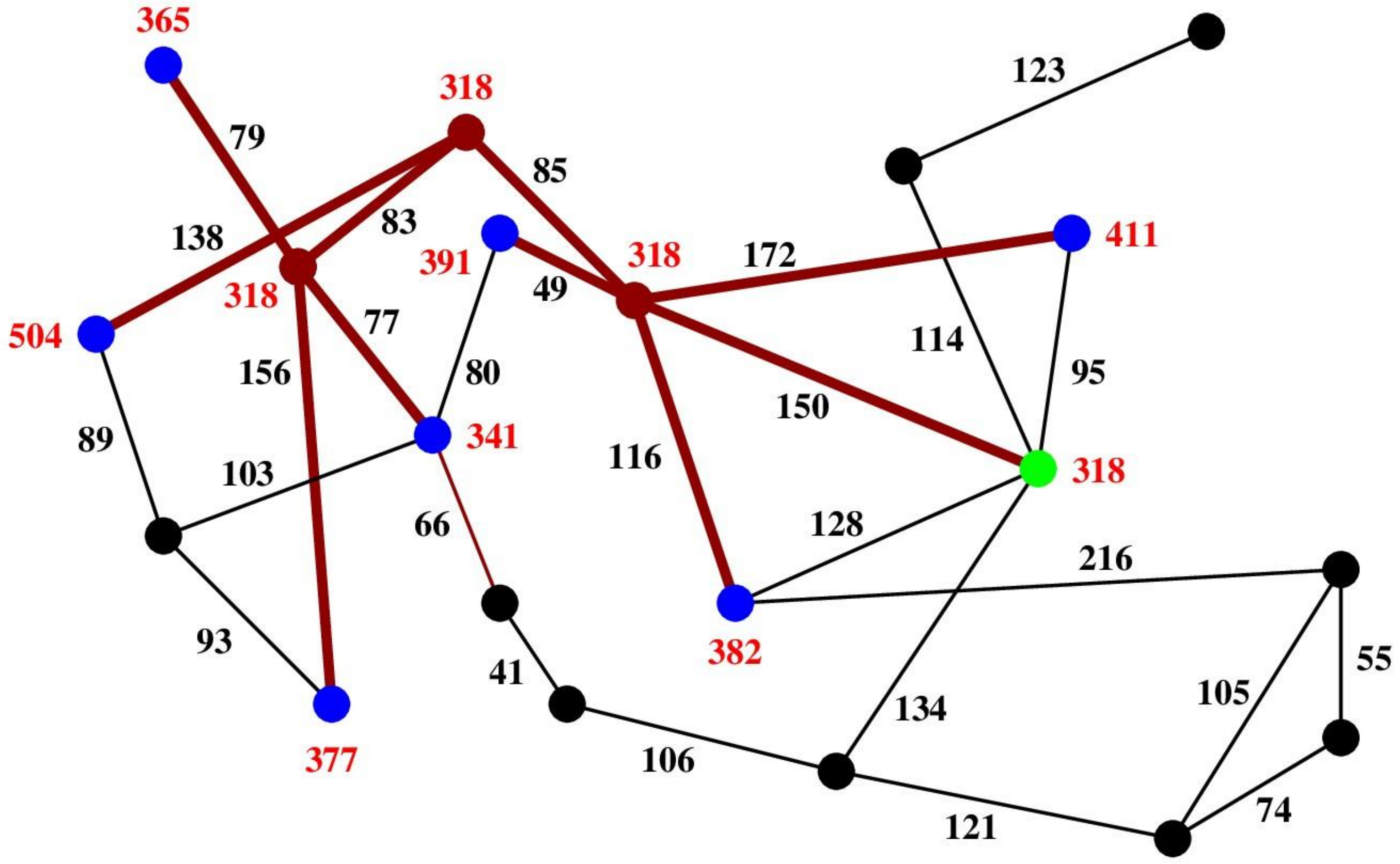
RBFS Example



RBFS Example



RBFS Example



Performance of RBFS

RBFS is cost-optimal if *h*() is an admissible heuristic.

RBFS may lead to many node re-expansions (forgotten trees may need to be regenerated).

Despite that, it performs better in general than IDA*.

A precise estimate of the running time of RBFS is difficult.

The space requirement for RBFS is O(*bD*), where *b* is the branching factor, and *D* is the depth of the/an optimal goal node.

Memory-bounded A* (MA*)

The memory requirements of IDA* and RBFS (also of IDS) are inconveniently low.

Even if enough memory is available, these algorithms are incapable of using that.

The price these algorithms pay is too many node re-generations.

PPC et al. propose the MA* algorithm in 1989.

MA* proceeds like A* using $f(n) = g(n) + h(n)$ until the memory usage hits a predefined bound.

When the memory becomes full, the open node with the largest $f()$ value is deleted.

Before this deletion, the $f()$ values of the ancestors of this open node are updated to record the effect of the (henceforth) missing leaf.

A simplified implementation of MA* is called **Simplified MA*** or **SMA*** (proposed by Russell in 1992).