Auction Algorithms for Path Planning, Network Transport, and Reinforcement Learning

by

Dimitri P. Bertsekas

Chapter 3
Auction Algorithms for Path Planning

This monograph represents "work in progress," and will be periodically updated. It more than likely contains errors (hopefully not serious ones). Furthermore, the references to the literature are incomplete. Your comments and suggestions to the author at dbertsek@asu.edu are welcome.

October 15, 2022

3

Auction Algorithms for Path Planning

Contents

In this chapter, we focus on path construction problems, including shortest path problems of various types. The starting point for our development is an algorithm, given in Section 3.1, which aims to compute some (not necessarily shortest) path from an origin to a destination node, and uses node prices to guide the search for the path. Algorithms of this type, and their relation to the auction algorithm for the assignment problem were discussed in Section 1.4, but in this chapter we undertake a more detailed development. Among others, our path construction algorithm will be used for constructing augmenting paths in the context of a max-flow algorithm in Section 4.2.

In Section 3.2, we will discuss extensions of the path construction algorithm of Section 3.1, which use arc lengths and produce a path that is "nearly shortest" with respect to these lengths (and shortest under some conditions). These extensions include the auction shortest path algorithm discussed in Section 1.4.3. We will discuss uses of our path construction algorithms for solving matching, assignment, max-flow, transportation, and transhipment problems in Chapter 4.

3.1 PATH CONSTRUCTION IN A DIRECTED GRAPH

In this section, we will introduce an algorithm, called *auction path construction* (APC for short), for finding a path from the origin node s to the destination node t in a directed graph, without aiming for any kind of optimality properties. The arcs of the graph are denoted by (i, j), where i and j are referred to as the *start* and *end* nodes of the arc. The sets of nodes and the set of arcs are denoted by $\mathcal{N}$ and $\mathcal{A}$, respectively. If (i, j) is an arc, it is possible that (j, i) is also an arc. No self arcs of the form (i, i) are allowed. We assume that for any two nodes i and j, there is at most one arc with start node i and end node j. For any node i we say that node j is a *downstream neighbor of* i if (i, j) is an arc.

The algorithm maintains a path starting at the origin, which at each iteration, is either *extended* by adding a new node, or *contracted* by deleting its terminal node. The decision to extend or contract is based on a set of variables, one for each node, which are called *prices*. Roughly speaking, the price of a node is viewed as a measure of the desirability of revisiting and advancing from that node in the future (low-price nodes are viewed as more desirable). Once the destination becomes the terminal node of the path, the algorithm terminates.

A node i is called *deadend* if it has no downstream neighbors. Note that s is not deadend, since we have assumed that there is a path from s to t.

Our algorithm maintains and updates a scalar price p_i for each node i. We say that under the current set of prices an arc (i, j) is:

(a) *Downhill*: If $p_i > p_j$.

(b) *Level*: If $p_i = p_j$.

(c) *Uphill*: If $p_i < p_j$.

Our algorithm also maintains and updates a directed path $P = (s, n_1, \ldots, n_k)$ that starts at the origin, and contains no cycles. The path is either the degenerate path $P = (s)$, or it ends at some node $n_k \neq s$, which is called the *terminal node* of P. If $P = (s)$, we also say that the terminal node of P is s.

Each iteration of the APC algorithm starts with a path and a price for each node, which are updated during the iteration using rules that we will now describe. The algorithm starts with the degenerate path $P = (s)$, and with some initial prices, which are arbitrary. † It terminates when a path has been found from s to t.

At each iteration when the algorithm starts with a path of the non-degenerate form $P = (s, n_1, \ldots, n_k)$, it either removes from P the terminal node n_k to obtain the new path $\overline{P} = (s, n_1, \ldots, n_{k-1})$, or it adds to P a node n_{k+1} to obtain the new path $\overline{P} = (s, n_1, \ldots, n_k, n_{k+1})$. In the former case the operation is called a *contraction* to n_{k-1}, and in the latter case it is called an *extension* to n_{k+1}.

At any one iteration the algorithm starts with a path P and a price p_i for each node i. At the end of the iteration a new path $\overline{P}$ is obtained from P through a contraction or an extension. Also the price of the terminal node of P [or the price p_s if $P = (s)$] is increased by a certain amount when there is a contraction. For iterations where the algorithm starts with the degenerate path $P = (s)$, only an extension is possible, i.e., $P = (s)$ is replaced by a path of the form $\overline{P} = (s, n_1)$.

A key feature of the algorithm, which in fact motivates its design, is that P and the prices p_i satisfy the following property at the start of each iteration for which $P \neq (s)$.

Downhill Path Property:
All arcs of the path $P = (s, n_1, \ldots, n_k)$ maintained by the APC algorithm are level or downhill. Moreover, the last arc (n_{k-1}, n_k) of P is downhill following an extension to n_k.

---

† The arbitrary nature of the initial prices is a major difference of our algorithm from earlier auction/path construction algorithms given in [Ber98], Section 2.6 and 3.3. Allowing arbitrary initial prices allows more flexibility in reusing prices from solution of one path finding problem to another similar problem. It also facilitates the use of "learned" prices that are favorable in similar problem contexts. This property can be important for computational efficiency in many applications.

The significance of the downhill path property is that *when an extension occurs, a cycle cannot be created*, in the sense that the terminal node n_k is different than all the predecessor nodes $s, n_1, \ldots, n_{k-1}$ on the path P. The reason is that the downhill path property implies that following an extension, we must have

$$p_{n_k} < p_{n_{k-1}} \leq p_{n_{k-2}} \leq \cdots \leq p_{n_1} \leq p_s,$$

showing that the terminal node n_k following an extension cannot be equal to any of the preceding nodes of P.

In addition to maintaining the downhill path property, the algorithm is structured so that following a contraction, which changes a nondegenerate path of the form $P = (s, n_1, \ldots, n_k)$ to $\overline{P} = (s, n_1, \ldots, n_{k-1})$, the price of n_k is increased by a positive amount. In conjunction with the fact that P never contains a cycle, this implies that either the algorithm terminates, or some node prices will increase to infinity. This is the key idea that underlies the validity of the algorithm, and forms the basis for its proof of termination.

To describe formally the algorithm, consider the case where $P \neq (s)$ and P has the form $P = (s, n_1, \ldots, n_k)$. We then denote by

$$\text{pred}(n_k) = n_{k-1}$$

the predecessor node of the terminal node n_k in the path P. [In the case where $P = (s, n_1)$, we use the notation $\text{pred}(n_1) = s$.] If the terminal node n_k of P is not deadend, we denote by $\text{succ}(n_k)$ a downstream neighbor of n_k that has minimal price:

$$\text{succ}(n_k) \in \arg \min_{\{j \mid (n_k, j) \in \mathcal{A}\}} p_j.$$

If multiple downstream neighbors of n_k have minimal price, the algorithm designates arbitrarily one of these neighbors as $\text{succ}(n_k)$.

The algorithm also uses a positive scalar ϵ. The choice of ϵ does not affect the path produced by the algorithm (so we could use $\epsilon = 1$ for example), but the choice of ϵ will play an important role in the weighted path construction algorithm of the next section.

The rules by which the path P and the prices p_i are updated at each

iteration are as follows; see Fig. 3.1.1.

Auction Path Construction Algorithm:

We distinguish three mutually exclusive cases.

(a) $P = (s)$: We then set the price p_s to $\max\{p_s, p_{\text{succ}(s)} + \epsilon\}$, and extend P to $\text{succ}(s)$.

(b) $P = (s, n_1, \ldots, n_k)$ *and node* n_k *is deadend*: We then set the price p_{n_k} to ∞ (or a very high number for practical purposes), and contract P to n_{k-1}.

(c) $P = (s, n_1, \ldots, n_k)$ *and node* n_k *is not deadend*. We then consider the following two cases.

(1) $p_{\text{pred}(n_k)} > p_{\text{succ}(n_k)}$. We then extend P to $\text{succ}(n_k)$ and set p_{n_k} to any price level that makes the arc $\big(\text{pred}(n_k), n_k\big)$ level or downhill and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill. [Setting

$$p_{n_k} = p_{\text{pred}(n_k)},$$

thus raising p_{n_k} to the maximum possible level, is a possibility, in which case the arc $\big(\text{pred}(n_k), n_k\big)$ becomes level; see Fig. 3.1.2.]

(2) $p_{\text{pred}(n_k)} \leq p_{\text{succ}(n_k)}$. We then contract P to $\text{pred}(n_k)$ and raise the price of n_k to the price of $\text{succ}(n_k)$ plus ϵ [thus making the arc $\big(\text{pred}(n_k), n_k\big)$ uphill and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill].

The algorithm terminates once the destination becomes the terminal node of P. We will show that eventually the algorithm terminates, under our standing assumption that there is at least one path from the origin to the destination.

The contraction/extension mechanism of the algorithm may be interpreted as a competitive process: we can view $\text{pred}(n_k)$ as being in competition with the downstream nodes of n_k for becoming the next terminal node of path P, after n_k. In particular, the terminal node of P moves to the node that offers minimal price [with ties that involve $\text{pred}(n_k)$ broken in favor of $\text{pred}(n_k)$ in order to maintain the downhill path property].

Figure 3.1.1 illustrates the extension and contraction mechanism of case (c) above, and shows how the downhill path property of the algorithm is maintained throughout its operation. In particular, the initial path $P = (s)$ satisfies the downhill path property trivially, since it contains no arcs. Furthermore, using Fig. 3.1.1 and the algorithm description, we can verify that if P and the node prices satisfy the downhill path property

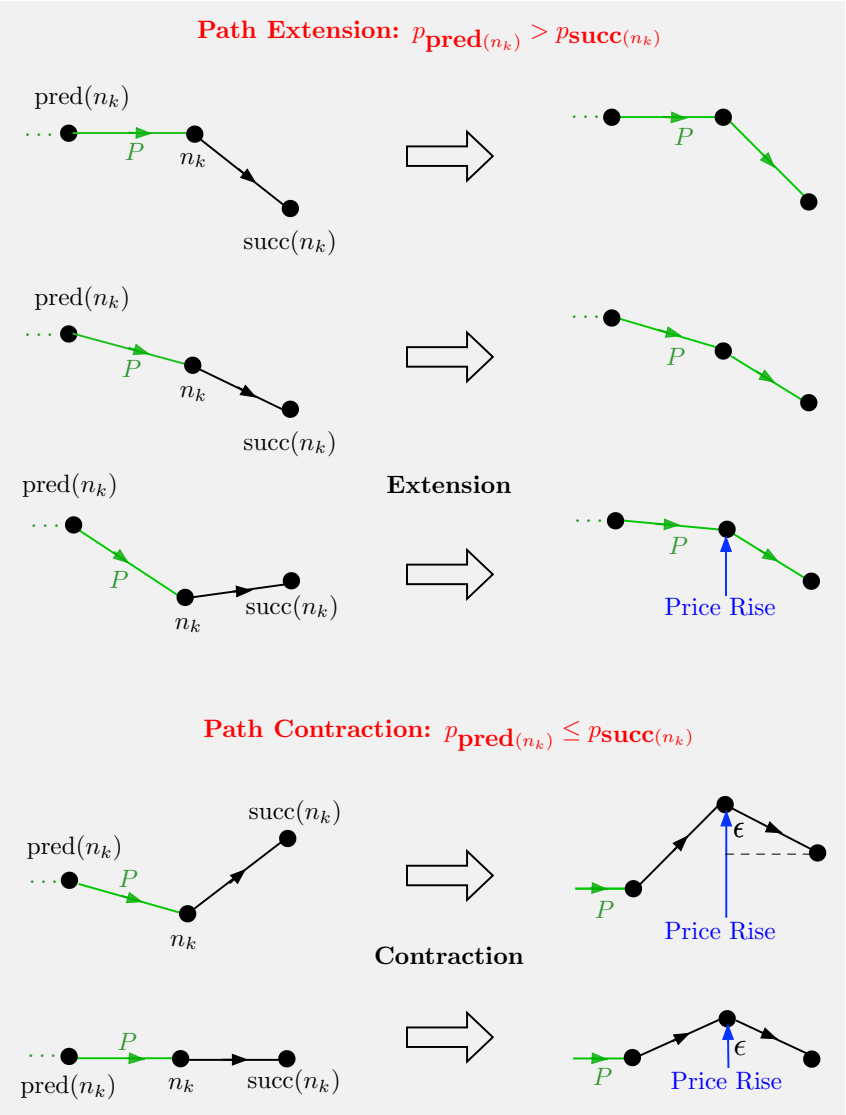


Figure 3.1.1 Illustration of the price levels of the terminal node n_k of the path $P = (s, n_1, \ldots, n_k)$, and the price levels of its predecessor and its successor, before and after an extension or a contraction; cf. cases (c1) and (c2) of the algorithm description. In the case where $p_{\text{pred}(n_k)} > p_{\text{succ}(n_k)}$, which corresponds to an extension, there may or may not be an increase of p_{n_k}. In the case where $p_{\text{pred}(n_k)} \leq p_{\text{succ}(n_k)}$, which corresponds to a contraction, there is always an increase of p_{n_k} by at least ϵ.

at the beginning of an iteration, then the new path and node prices at the beginning of the next iteration also satisfy the downhill path property. Fig-

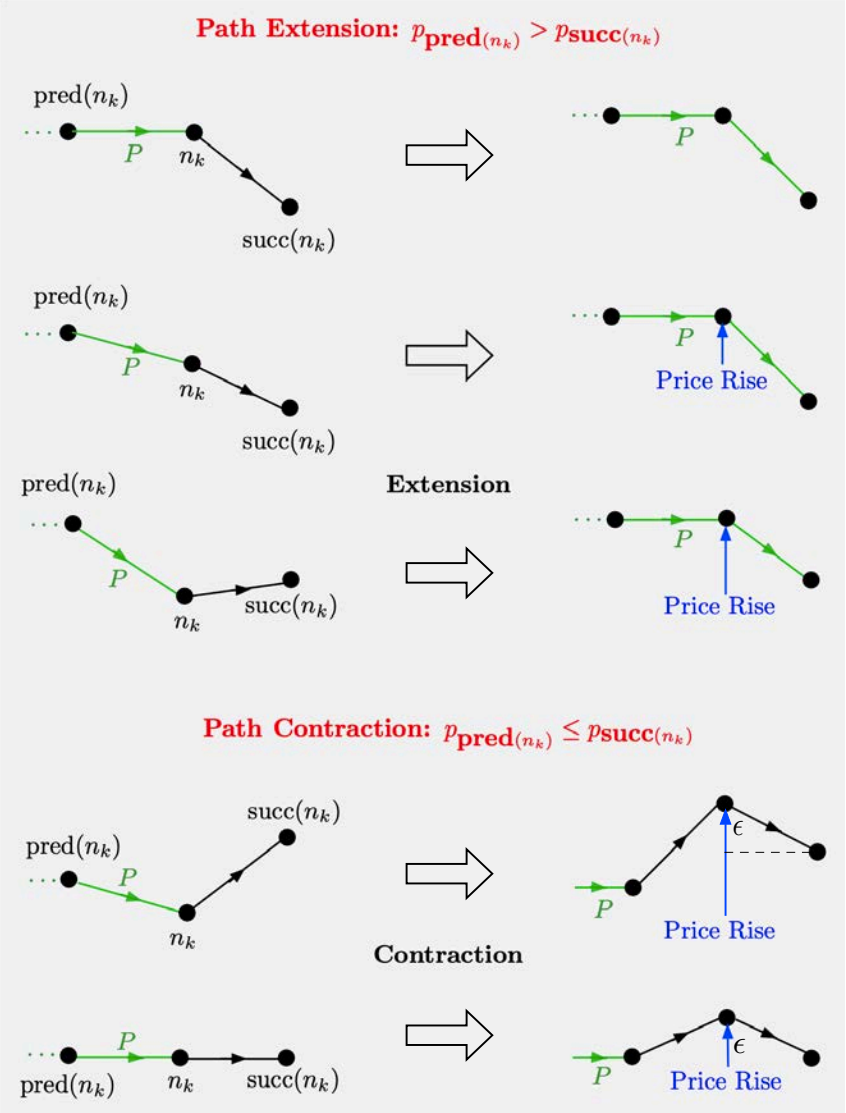


Figure 3.1.2 Illustration of the special case of the APC algorithm that sets $p_{n_k} = p_{\text{pred}(n_k)}$, raising p_{n_k} to the maximum possible level $p_{n_k} = p_{\text{pred}(n_k)}$ in the case of the extension step (c1). In this case the predecessor arc $\big(\text{pred}(n_k), n_k\big)$ is forced to become level, and the arcs of path P are all level, except for the last arc following an extension.

ure 3.1.2 illustrates the extension and contraction mechanism in the special case where p_{n_k} is raised to the maximum possible level $p_{n_k} = p_{\text{pred}(n_k)}$ following an extension, thus making the predecessor arc $\big(\text{pred}(n_k), n_k\big)$ level.

3.1.1 Algorithm Justification

We will prove that eventually the destination will become the terminal node of P, at which time the algorithm will terminate. To this end we argue by contradiction and we use our assumption that there is at least one path from the origin to the destination.

Indeed, suppose, to arrive at a contradiction, that the algorithm does not terminate. Then, since the path P does not contain a cycle and hence cannot extend indefinitely, the algorithm must perform an infinite number of contractions. Let $\mathcal{N}_\infty$ be the nonempty set of nodes whose price increases (by at least ϵ) infinitely often due to a contraction (and hence their price increases to ∞). Let also $\overline{\mathcal{N}}_\infty = \{i \mid i \notin \mathcal{N}_\infty\}$ be the complementary set of nodes whose price increases due to a contraction finitely often (and hence do not become the terminal node of P after some iteration). Clearly, by the rules of the algorithm, there is no arc connecting a node of $\mathcal{N}_\infty$ to a node of $\overline{\mathcal{N}}_\infty$. Moreover, the destination t clearly belongs to $\overline{\mathcal{N}}_\infty$, and we claim that the origin s belongs to $\mathcal{N}_\infty$. Indeed, if $s \in \overline{\mathcal{N}}_\infty$ there would exist a subpath $P' = (s, n_1, \ldots, n_k)$ such that the nodes $s, n_1, \ldots, n_{k-1}$ belong to $\overline{\mathcal{N}}_\infty$, the last node n_k belongs to $\mathcal{N}_\infty$, and P' is the initial portion of P for all iterations after finitely many. Since n_k will be the terminal node of P infinitely often, it follows that n_{k-1} will be the predecessor $\text{pred}(n_k)$ of n_k infinitely often, while the price of n_k increases to infinity and the price of n_{k-1} stays finite. By the rules of the algorithm, this is not possible. Thus we must have $s \in \mathcal{N}_\infty$, $t \in \overline{\mathcal{N}}_\infty$, and no arc connecting a node of $\mathcal{N}_\infty$ to a node of $\overline{\mathcal{N}}_\infty$. This contradicts the assumption that there is a path from s to t, and shows that the algorithm will terminate.

We summarize the preceding arguments in the following proposition.

Proposition 3.1.1: If there exists at least one path from the origin to the destination, the APC algorithm terminates with a path from s to t. Otherwise the algorithm never terminates and we have $p_i \to \infty$ for all nodes i in a subset $\mathcal{N}_\infty$ that contains s.

3.2 WEIGHTED PATH CONSTRUCTION

We will now consider the shortest path problem, which involves a length a_{ij} for each arc (i, j). We consider a generalization of our path construction algorithm, the *auction weighted path construction* algorithm (AWPC for short), which we have also discussed briefly in Section 1.4. The arc lengths serve to provide a bias towards producing paths with small total length. In fact in many cases (but not always) the algorithm produces shortest

paths with respect to the given lengths. We require that *all cycles have nonnegative length*. By this we mean that for every cycle $(i, n_1, \ldots, n_k, i)$ we have

$$a_{i,n_1} + a_{n_1 n_2} + \cdots + a_{n_{k-1} n_k} + a_{n_k i} \geq 0. \tag{3.1}$$

This is a common assumption in shortest path problems.

Extending the terminology of the preceding section, we say that under the current set of prices and lengths an arc (i, j) is:

(a) *Downhill*: If $p_i > a_{ij} + p_j$.

(b) *Level*: If $p_i = a_{ij} + p_j$.

(c) *Uphill*: If $p_i < a_{ij} + p_j$.

We denote by

$$\text{pred}(n_k) = n_{k-1}$$

the predecessor node of the terminal node n_k in the path P. [In the case where $P = (s, n_1)$, we let $\text{pred}(n_1) = s$.] If the terminal node n_k of P is not deadend, we denote by $\text{succ}(n_k)$ a downstream neighbor j of n_k for which $a_{n_k j} + p_j$ is minimized:

$$\text{succ}(n_k) \in \arg \min_{\{j \mid (n_k, j) \in \mathcal{A}\}} \{a_{n_k j} + p_j\}.$$

If multiple downstream neighbors of n_k attain the minimum, the algorithm designates arbitrarily one of these neighbors as $\text{succ}(n_k)$.

The AWPC algorithm, introduced in Section 1.4, maintains and updates a directed path $P = (s, n_1, \ldots, n_k)$ and a price p_i for each node i. The path is either the degenerate path $P = (s)$, or it ends at some node $n_k \neq s$, which, as earlier, is called the *terminal node* of P. Each iteration starts with a path and a price for each node, which are updated during the iteration. The algorithm starts with the degenerate path $P = (s)$, and the initial prices are arbitrary. It terminates when the destination becomes the terminal node of P.

We will now describe the rules by which the path and the prices are updated. At any one iteration the algorithm starts with a path P and a scalar price p_i for each node i. At the end of the iteration a new path $\overline{P}$ is obtained from P through a contraction or an extension as earlier. For iterations where the algorithm starts with the degenerate path $P = (s)$, only an extension is possible, i.e., $P = (s)$ is replaced by a path of the form $\overline{P} = (s, n_1)$. Also the price of the terminal node of P is increased just before a contraction, and in some cases, just before an extension. The amount of price rise is determined by a scalar parameter $\epsilon > 0$.

The algorithm terminates when the destination becomes the terminal node of P. The rules by which the path P and the prices p_i are updated at every iteration prior to termination are as follows.

Auction Weighted Path Construction Iteration:

We distinguish three mutually exclusive cases.

(a) $P = (s)$: We then set the price p_s to $\max\{p_s, a_s\text{succ}(s) + p_{\text{succ}(s)} + \epsilon\}$, and extend P to $\text{succ}(s)$.

(b) $P = (s, n_1, \ldots, n_k)$ *and node* n_k *is deadend*: We then set the price p_{n_k} to ∞ (or a very high number for practical purposes), and contract P to n_{k-1}.

(c) $P = (s, n_1, \ldots, n_k)$ *and node* n_k *is not deadend*. We consider the following two cases.

(1) $p_{\text{pred}(n_k)} > a_{\text{pred}(n_k)n_k} + a_{n_k}\text{succ}(n_k) + p_{\text{succ}(n_k)}$. We then extend P to $\text{succ}(n_k)$ and set p_{n_k} to any price level that makes the arc $\big(\text{pred}(n_k), n_k\big)$ level or downhill and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill. [Setting

$$p_{n_k} = p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k},$$

thus raising p_{n_k} to the maximum possible level, is a possibility, in which case the arc $\big(\text{pred}(n_k), n_k\big)$ becomes level; cf. Fig. 3.1.2.]

(2) $p_{\text{pred}(n_k)} \leq a_{\text{pred}(n_k)n_k} + a_{n_k}\text{succ}(n_k) + p_{\text{succ}(n_k)}$. We then contract P to $\text{pred}(n_k)$ and raise the price of n_k to

$$a_{n_k}\text{succ}(n_k) + p_{\text{succ}(n_k)} + \epsilon$$

[thus making the arcs $\big(\text{pred}(n_k), n_k\big)$l and $\big(n_k, \text{succ}(n_k)\big)$ uphill and downhill, respectively].

A downhill/level/uphill type of interpretation, similar to Fig. 3.1.1, applies to this algorithm as well [the relative heights of the prices of nodes $\text{pred}(n_k)$, $\text{succ}(n_k)$, and n_k, indicated in Fig. 3.1.1 should incorporate the arc lengths $a_{\text{pred}(n_k)n_k}$ and $a_{n_k}\text{succ}(n_k)$, as in the preceding algorithm description]. There is a price increase of n_k in the case of a contraction, and also in the case of an extension if the arc $\big(\text{pred}(n_k), n_k\big)$ is downhill. However, the conditions for an arc (i, j) to be downhill, level, or uphill involve the arc lengths a_{ij}. Similar to our earlier arguments, it can be seen that P and the prices p_i satisfy the following downhill path property at the start of each iteration for which $P \neq (s)$.

Downhill Path Property:
All arcs of the path $P = (s, n_1, \ldots, n_k)$ maintained by the AWPC algorithm are level or downhill. Moreover, the last arc (n_{k-1}, n_k) of P is downhill following an extension to n_k.

A consequence of this property (and our assumption that all cycles have nonnegative length) is that when an extension occurs, a cycle cannot be created, in the sense that the terminal node n_k is different than all the predecessor nodes $s, n_1, \ldots, n_{k-1}$ on the path $P = (s, n_1, \ldots, n_k)$. Thus, assuming that there is at least one path from the origin to the destination, it can be shown that eventually the destination will become the terminal node of P, at which time the algorithm will terminate. The proof is essentially identical to the proof we gave earlier for the case of zero lengths in Prop. 3.1.1. For an illustration of the algorithm, see the example given in Section 1.4.

Proposition 3.2.1: If there exists at least one path from the origin to the destination and the nonnegative cycle condition (3.1) holds, the AWPC algorithm terminates with a path from s to t. Otherwise the algorithm never terminates and we have $p_i \to \infty$ for all nodes i in a subset $\mathcal{N}_\infty$ that contains s.

Note that the proposition does not guarantee that the final path generated by the AWPC algorithm is shortest. The deviation from optimality of the algorithm depends on the initial prices, as well as the parameter ϵ. We will quantify this dependence in what follows.

3.2.1 The Role of the Parameter ϵ - Convergence Rate and Solution Accuracy Tradeoff

In auction algorithms, it is common to use a positive ϵ parameter to regulate the size of price rises. In the AWPC algorithm, ϵ is used to provide an important tradeoff between the ability of the algorithm to construct paths with near-minimum length, and its rate of convergence. Generally, as ϵ becomes smaller the quality of the path produced improves, as we will show with examples and analysis in what follows. On the other hand a small value of ϵ tends to slow down the algorithm.

In what follows in this section, we will use two examples to illustrate how the choice of ϵ affects the rate of convergence of the AWPC algorithm,

as well as the error from optimality of the path that it produces. For both examples, in the extension case (c1), we set the price level of n_k to the maximum possible level,

$$p_{n_k} = p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k},$$

cf. Fig. 3.1.2.

Example 3.2.1 (Nonpolynomial Behavior and ϵ-Scaling)

This is an example of a shortest path problem where there is a cycle of relatively small length. It involves that graph of the top figure in Fig. 3.2.2. The cycle consists of nodes 1, 2, and 3, and has length 0 [the algorithm's behavior is similar when the cycle has positive length that is small relative to L, the length of the last arc of the unique s-to-t path]. Such cycles slow down the algorithm, when ϵ has a small value. Indeed, it can be seen from the table of Fig. 3.2.2 that for small values of ϵ and initial prices equal to 0, the algorithm repeats the cycle

$$s \to 1 \to 2 \to 3 \to 2 \to 1 \to s \to 1 \cdots$$

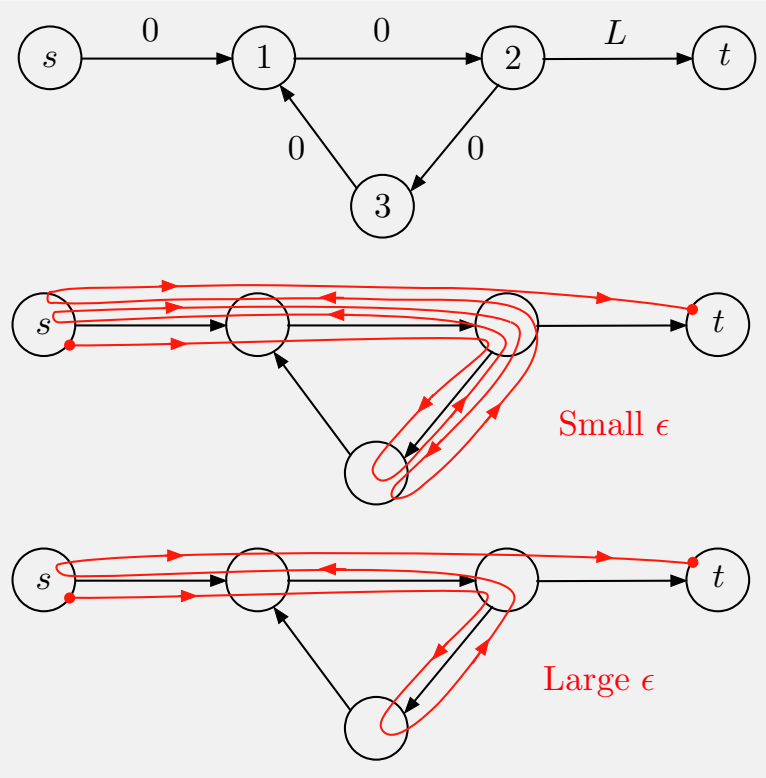
until the prices of nodes 2 and 3 reach levels $p_2 > L$ and $p_3 > L$, so that the arc $(2, t)$ becomes downhill and an extension from 2 to t is performed. The number of cycles for this to happen depends on ϵ and is roughly proportional to L/ϵ, so for small values of ϵ, the computation is nonpolynomial (see the middle part of the figure). On the other hand, it can be seen from the table of Fig. 3.2.2 that when ϵ is large enough so that $3 + 4\epsilon > L$, the algorithm moves to t once it reaches node 2 for the second time, after 8 iterations.

It can be shown that with an ϵ-scaling scheme, whereby ϵ is reduced by a certain factor between successive runs of the algorithm, the computation becomes polynomial, proportional to $\log L$, rather than L. This is a common property of auction algorithms; see the book [Ber98] and the references quoted there. Later we will discuss the use of ϵ-scaling and its use to provide convergence acceleration as well as exact shortest path solutions.

In the preceding example, the poor performance of the algorithm is caused by the presence of a cycle with small length. The next example illustrates how a similar phenomenon can also occur in acyclic graphs involving many-node paths.

Example 3.2.2 (Convergence Rate and Solution Accuracy Tradeoff)

Consider a graph involving a long chain of nodes that starts at the origin and ends at the destination, as shown in Fig. 3.2.3. We assume that the initial prices are all equal to 0. Then it can be verified that for large values of ϵ, the algorithm will terminate with the suboptimal path $(s, 1, 2, \ldots, n, t)$; in fact for $\epsilon > n$, it will terminate in $n + 1$ iterations through the sequence of extensions



Iteration #	Path P prior to iteration	Prices prior to iteration	Type of iteration
1	(s)	$(\underline{0}, 0, 0, 0, 0)$	Extension to 1
2	$(s, 1)$	$(\epsilon, \underline{0}, 0, 0, 0)$	Extension to 2
3	$(s, 1, 2)$	$(\epsilon, \epsilon, \underline{0}, 0, 0)$	Extension to 3
4	$(s, 1, 2, 3)$	$(\epsilon, \epsilon, \epsilon, \underline{0}, 0)$	Contraction to 2
5	$(s, 1, 2)$	$(\epsilon, \epsilon, \underline{\epsilon}, 2\epsilon, 0)$	Contraction to 1
6	$(s, 1)$	$(\epsilon, \underline{\epsilon}, 3\epsilon, 2\epsilon, 0)$	Contraction to s
7	(s)	$(\underline{\epsilon}, 4\epsilon, 3\epsilon, 2\epsilon, 0)$	Extension to 1
8	$(s, 1)$	$(5\epsilon, \underline{4\epsilon}, 3\epsilon, 2\epsilon, 0)$	Extension to 2
9	$(s, 1, 2)$	$(5\epsilon, 5\epsilon, \underline{3\epsilon}, 2\epsilon, 0)$	Extension to t if $2\epsilon > L$
			Extension to 2 otherwise
...	...	...	Continue until $p_3 > L$

Figure 3.2.2 The shortest path problem of Example 3.2.1 (top part of the figure). The arc lengths are shown next to the arcs [all lengths are equal to 0, except for the length of arc $(2, t)$ which has a large length L]. There is only one point where the algorithm can go wrong, at node 2 where there is a choice between going to t or going to 3. The only s-to-t path is $(s, 1, 2, t)$, but if ϵ is very small, the algorithm explores the possibility of reaching the destination through node 3 for many iterations, while repeating the cycle $s \to 1 \to 2 \to 3 \to 2 \to 1 \to s \to 1 \ldots$ (middle part of the figure). On the other hand, if $2\epsilon > L$, then at iteration 9, following an extension to node 2, the successor to node 2 is t, the algorithm compares the prices of nodes 3 and t, performs an extension to t, and terminates (bottom part of the figure).

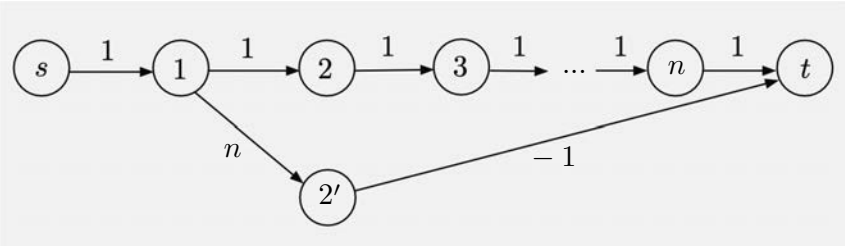


Figure 3.2.3 The graph of the shortest path problem of Example 3.2.2. The arc lengths are shown next to the arcs. All lengths are equal to 1, except for the length of arc $(2', t)$ which is equal to -1 and the length of arc $(1, 2')$ which is equal to n, the number of intermediate nodes in the top path from s to t. The shortest path is $(s, 2', t)$, but for large values of ϵ, the algorithm will terminate with the suboptimal path $(s, 1, 2, \ldots, n, t)$.

$s \to 1 \to 2 \to \cdots \to n \to t$. It can also be verified, by tracing the steps of the algorithm that for small values of ϵ, the algorithm will find the optimal path $(s, 2', t)$, but will need a large number of iterations (proportional to n^2) to do so. A suitable ϵ-scaling scheme can find the optimal path in $O(n \log n)$ iterations.

3.2.2 Shortest Distances and Error Bounds

An important issue in the AWPC algorithm is the choice of the initial prices. We will argue that the algorithm operates effectively, in the sense that it terminates fast and with small error from path optimality, if the initial prices are close to the shortest distances under the given set of lengths. Indeed, the lengths $\{a_{ij}\}$ define the shortest distances, denoted by D_i^*, from the nodes i to the destination t. These shortest distances satisfy $D_t^* = 0$ and for all $i \neq t$,

$$D_i^* = \min_{\{j \mid (i,j) \in \mathcal{A}\}} \{a_{ij} + D_j^*\};$$

this is an instance of the fundamental dynamic programming/Bellman equation. It implies that all arcs are level or uphill, with respects to prices $p_i = D_i^*$, and the arcs of a shortest path are level. Suppose that we choose for all i a price p_i that is exactly equal to D_i^*. Then it can be verified that starting from an arbitrary origin i, the algorithm generates a shortest path from i to t through a sequence of extensions over level arcs, without any intervening contractions. The price differential $p_s - p_t$ is equal to the total length of the path produced by the algorithm, which is shortest.

If the initial prices p_i are not equal to the shortest distances D_i^*, the price differential $p_s - p_t$ provides an upper bound to the total length L_P of the final path P produced by the algorithm:

$$L_P = \sum_{(i,j) \in P} a_{ij} \leq p_s - p_t. \tag{3.2}$$

To see this, note that for $P = (s, n_1, \ldots, n_k)$ we have $a_{sn_1} = p_s - p_{n_1}$, $a_{n_{i-1}n_i} = p_{n_{i-1}} - p_{n_i}$ for all $i = 1, \ldots, k$, and $a_{n_k t} \leq p_{n_k} - p_t$, since in view of the downhill property, all the arcs of P are level or downhill. It follows that

$$\begin{aligned} L_P &= a_{sn_1} + a_{n_1 n_2} + \cdots + a_{n_{k-1}n_k} + a_{n_k t} \\ &\leq (p_s - p_{n_1}) + (p_{n_1} - p_{n_2}) + \cdots + (p_{n_{k-1}} - p_{n_k}) + (p_{n_k} - p_t) \\ &= p_s - p_t, \end{aligned}$$

thus verifying Eq. (3.2).

If all the arcs that do not belong to P are level or uphill upon termination, then it can be shown that P has minimum total length. More generally, we will show an error bound that involves the amounts d_{ij} by which a_{ij} must be increased to make the arc (i, j) level if it is downhill:

$$d_{ij} = \max\{0, p_i - a_{ij} - p_j\}, \qquad (i, j) \in \mathcal{A}. \tag{3.3}$$

The scalars d_{ij} will be referred to as the *discrepancies* of the arcs (i, j). They quantify the error from optimality of the path P generated by the algorithm, as shown in the following proposition.

Proposition 3.2.2: Let the AWPC algorithm terminate with a path P, and let P' be any other path from s to t. Then we have

$$L_P + \sum_{(i,j)\in P} d_{ij} \leq L_{P'} + \sum_{(i,j)\in P'} d_{ij}, \tag{3.4}$$

where L_P and $L_{P'}$ are the total lengths of P and P', and d_{ij} are the arc discrepancies of Eq. (3.3), which are obtained upon termination.

Proof: Suppose that we increase the arc lengths a_{ij} by the corresponding arc discrepancies d_{ij} that are obtained upon termination, thus changing these lengths to

$$\bar{a}_{ij} = a_{ij} + d_{ij}, \qquad (i, j) \in \mathcal{A}.$$

Then upon termination, the path P produced by the AWPC algorithm is shortest with respect to arc lengths $\bar{a}_{ij}$. The reason is that the arcs that belong to P are level with respect to the arc lengths $\bar{a}_{ij}$, while the arcs that do not belong to P are either level or uphill, again with respect to $\bar{a}_{ij}$; this is the optimality condition for P to be shortest with respect to $\bar{a}_{ij}$. The inequality (3.4) then follows, since its left and right sides are the lengths of P and P', respectively, with respect to $\bar{a}_{ij}$. **Q.E.D.**

The discrepancies d_{ij} provide an upper bound to the degree of suboptimality of the path obtained upon termination, as per Eq. (3.4). In particular, if $\overline{D}$ is the maximum arc discrepancy upon termination of the algorithm,

$$\overline{D} = \max_{(i,j)\in\mathcal{A}} d_{ij},$$

then, since $d_{ij} \geq 0$ for all (i,j), Eq. (3.4) implies that

$$L_P \leq L_{P'} + (n+1)\overline{D}, \tag{3.5}$$

where n is the number of nodes other than s and t. The reason is that a path from s to t can contain at most $(n+1)$ arcs, each having a discrepancy that is at most $\overline{D}$.

An interesting empirical observation is that when the algorithm creates a new downhill arc (i,j) that lies outside P, the corresponding discrepancy d_{ij} becomes equal to ϵ or a small multiple of ϵ. A reasonable conjecture is that if all the discrepancies d_{ij} are initially bounded by a small multiple of ϵ, then the path produced by the algorithm upon termination is shortest to within a small multiple of $n\epsilon$, where n is the number of nodes. This bears similarity to auction algorithms for assignment and network flow problems, where the solution obtained can be proved to be optimal to within $n\epsilon$.

3.2.3 ϵ-Complementary Slackness - Using ϵ-Scaling to Find a Shortest Path

The tradeoff between speed of convergence and accuracy of solution that is embodied in the choice of ϵ was recognized in the original proposal of the auction algorithm for the assignment problem [Ber79], and the approach of *ϵ-scaling* was proposed to deal with it. In this approach we start the auction algorithm with a relatively large value of ϵ, to obtain quickly rough estimates for appropriate values of the node prices, and then we progressively reduce ϵ to refine the node prices and eventually obtain an optimal solution. The use of ϵ-scaling also allows the option of stopping the algorithm, with a less refined solution, if the allotted time for computation is limited.

In the context of the AWPC algorithm, ϵ-scaling is implemented by running the algorithm with a relatively large value of ϵ to estimate "good" prices, at least for a subset of "promising" transit nodes from s to t, and then progressively refining the assessment of the "promise" of these nodes. This is done by rerunning the algorithm with smaller values of ϵ, while using as initial prices at each run the final prices of the previous run. It is well-known that ϵ-scaling improves the computational complexity of auction

algorithms,† and it can be similarly applied to the AWPC algorithm, as we will discuss in Section 3.3.

Empirically, this scheme seems to work well, but it does not offer a guarantee that it will yield a shortest path. As an example in the problem of Fig. 3.2.3, if all prices are chosen to be 0 except for the price of node 2', which is chosen to be a high positive number, the AWPC algorithm will not find the shortest path for any value of ϵ.

We will now consider a variant of the AWPC algorithm and corresponding ϵ-scaling scheme that guarantee that a shortest path can be obtained. The ϵ-scaling scheme modifies the prices produced by the AWPC algorithm for a given value of ϵ, before applying the algorithm with a smaller value of ϵ. To this end, we will operate the AWPC algorithm so that it initially satisfies and subsequently maintains the following *ϵ-complementary slackness* condition (ϵ-CS for short).

ϵ-Complementary Slackness:

For a given $\epsilon > 0$, the prices $\{p_i \mid i \in \mathcal{N}\}$ and the path P satisfy

$$p_i \leq a_{ij} + p_j + \epsilon, \qquad \text{for all arcs } (i,j),$$

i.e., every arc is uphill, or level, or downhill by at most ϵ, and

$$p_i \geq a_{ij} + p_j, \qquad \text{for all arcs } (i,j) \text{ of the path } P,$$

i.e., every arc of P is level or downhill (by at most ϵ).

The notion of ϵ-CS is fundamental in the context of auction algorithms, and represents a relaxation of the classical complementary slackness condition of linear programming (see, e.g., Bertsimas and Tsitsiklis [BeT97]). In particular, when ϵ-CS holds, the discrepancies d_{ij} of Eq. (3.3) are at most equal to ϵ, so *if the AWPC algorithm maintains ϵ-CS throughout its operation, it produces a path that is suboptimal by at most* $(n+1)\epsilon$ in view of Eq. (3.5), and hence also optimal for ϵ sufficiently small [$(n+1)\epsilon$ should be less than the difference between the 2nd shortest path distance and the shortest path distance]. Thus maintaining ϵ-CS is desirable.

On the other hand, the AWPC algorithm need not maintain ϵ-CS throughout its operation, because the increase of p_{n_k} prior to an extension may violate the ϵ-CS inequality $p_{n_k} \leq a_{n_k j} + p_j + \epsilon$ for $j = \text{succ}(n_k)$ and possibly for j equal to some other downstream neighbors of n_k. A simple

† See the papers [BeE88], [Ber88], the book [Ber98], and the references quoted there, for polynomial complexity analyses of auction algorithms for the assignment problem and other related problems.

remedy is to choose the price increase prior to an extension in a specific way. In particular, in case (c1) of the AWPC algorithm, we raise the price p_{n_k} to the largest value that satisfies ϵ-CS, while extending P to $\text{succ}(n_k)$, rather than setting p_{n_k} to any value that makes the arc $\big(\text{pred}(n_k), n_k\big)$ level or downhill and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill.

More specifically, there are three cases to consider when the relation

$$p_{\text{pred}(n_k)} > a_{\text{pred}(n_k)n_k} + a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)}, \tag{3.6}$$

holds, which are illustrated in Fig. 3.2.4:

(a) **The arc $\big(\textbf{pred}(n_k), n_k\big)$ is level**, i.e.,

$$0 = p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k} - p_{n_k},$$

$$0 < p_{n_k} - a_{n_k\text{succ}(n_k)} - p_{\text{succ}(n_k)}.$$

Then we extend to $\text{succ}(n_k)$ and leave the price p_{n_k} unchanged (top part of Fig. 3.2.4).

(b) **The arc $\big(\textbf{pred}(n_k), n_k\big)$ is downhill and the arc $\big(n_k, \textbf{succ}(n_k)\big)$ is downhill or level**, i.e.,

$$0 < p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k} - p_{n_k},$$

$$0 \leq p_{n_k} - a_{n_k\text{succ}(n_k)} - p_{\text{succ}(n_k)}.$$

Then we extend to $\text{succ}(n_k)$ and set the price p_{n_k} to

$$\min\big\{p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k},\, a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)} + \epsilon\big\},$$

thus making the arc $\big(\text{pred}(n_k), n_k\big)$ is downhill or level and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill (middle part of Fig. 3.2.4).

(c) **The arc $\big(\textbf{pred}(n_k), n_k\big)$ is downhill and the arc $\big(n_k, \textbf{succ}(n_k)\big)$ is uphill**, i.e.,

$$0 < p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k} - p_{n_k},$$

$$p_{n_k} - a_{n_k\text{succ}(n_k)} - p_{\text{succ}(n_k)} < 0.$$

Then we extend to $\text{succ}(n_k)$ and set the price p_{n_k} to

$$p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k},$$

thus making the arc $\big(\text{pred}(n_k), n_k\big)$ level (if ϵ-CS is satisfied on this arc) or downhill (if ϵ-CS is violated on this arc), and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill (bottom part of Fig. 3.2.4).

We refer to the AWPC algorithm with the specific price change rule given above as the *AWPC-CS algorithm*. It is a special case of the AWPC algorithm, and therefore it maintains the downhill path property, thus guaranteeing termination. It also ensures that the ϵ-CS property holds upon termination, provided the initial price discrepancies are bounded by ϵ. As a result the path obtained upon termination is shortest to within $(n+1)\epsilon$; cf. Eq. (3.5).

We summarize the preceding arguments in the following proposition.

Proposition 3.2.3: Assume that there exists at least one path from the origin to the destination, and the nonnegative cycle condition (3.1) holds. Then the AWPC-CS algorithm terminates with a path from s to t. If in addition the initial prices satisfy ϵ-CS, then the algorithm will maintain the ϵ-CS property throughout its operation, and the path obtained termination will be shortest to within $(n+1)\epsilon$, where n is the number of nodes other than s and t. Otherwise the algorithm will never terminate and we have $p_i \to \infty$ for all nodes i in a subset $\mathcal{N}_\infty$ that contains s.

When the starting prices violate ϵ-CS for some arcs, it is possible that the ϵ-CS property is accidentally restored at some point during the operation of the AWPC-CS algorithm, in which case ϵ-CS will hold upon termination as per the preceding discussion. However, there is no guarantee that this will happen. On the other hand, as Fig. 3.2.4 illustrates, the AWPC-CS algorithm does not create new arcs that violate ϵ-CS. Thus *the algorithm has a tendency to self-correct*. In fact the following property of the algorithm can be verified: *the maximum arc violation of ϵ-CS, as measured by*

$$\overline{D}_\epsilon = \max_{(i,j)\in\mathcal{A}} \max\{0,\, p_i - a_{ij} - p_j - \epsilon\},$$

is not increased at any iteration.

Implementing ϵ-Scaling

Given the final set of prices and path obtained by the AWPC-CS algorithm for a given value of ϵ, there is an important issue in ϵ-scaling: how to modify the prices of some of the nodes so that the resulting prices together with the degenerate path (s) satisfy ϵ'-CS for a smaller positive value $\epsilon' < \epsilon$. Moreover the price modifications should be small in order for the new prices to be good starting points for rerunning the algorithm with the new value ϵ'.

There are algorithms for computing price modifications to satisfy ϵ'-CS for a smaller value $\epsilon' < \epsilon$ together with the degenerate path (s), which

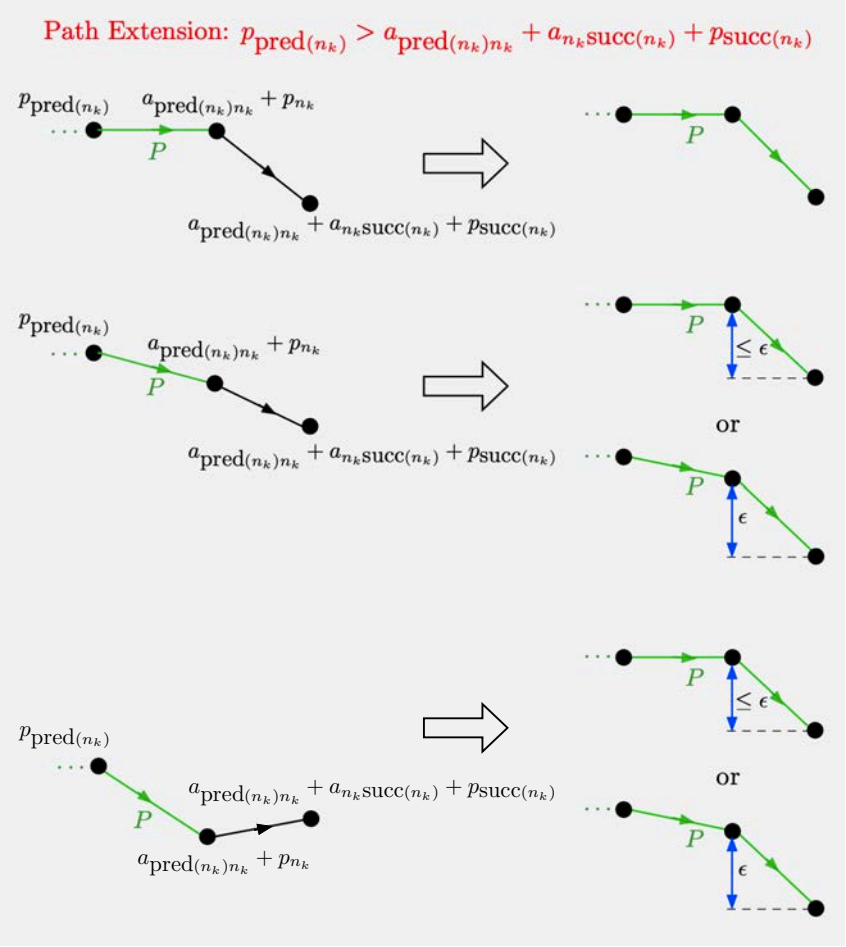


Figure 3.2.4 Illustration of an iteration of the AWPC-CS algorithm, which maintains ϵ-CS if started with prices satisfying ϵ-CS. The figure shows the levels

$$p_{\text{pred}(n_k)}, \qquad a_{\text{pred}(n_k)n_k} + p_{n_k}, \qquad a_{\text{pred}(n_k)n_k} + a_{n_k}\text{succ}(n_k) + p_{\text{succ}(n_k)},$$

before and after an extension. The middle and bottom portions of the figure illustrate how following the extension, it is possible that both arcs $\big(\text{pred}(n_k), n_k\big)$ and $\big(n_k, \text{succ}(n_k)\big)$ are downhill. In the case where ϵ-CS is satisfied on the arc $\big(\text{pred}(n_k), n_k\big)$, the arc will become level following the price rise.

will be discussed in future reports. Moreover, often such algorithms can take advantage of special structure of the problem's graph. This is true for example in assignment problems, where the bipartite character of the graph allows great flexibility in the choice of the initial prices. In what fol-

lows in this section, we discuss the case of shortest path problems involving an acyclic graph, which arise prominently in on-line and off-line multistep lookahead minimization, and tree search for reinforcement learning problems (see Section 3.5).

ϵ-Scaling in Acyclic Graphs

Let us consider the case of an acyclic graph; cf. Fig. 3.2.5. Suppose that we are given arc lengths a_{ij} and a set of prices $\{p_i \mid i \in \mathcal{N}\}$ that for a given positive ϵ satisfy

$$p_i \leq a_{ij} + p_j + \epsilon, \qquad \text{for all arcs } (i,j), \tag{3.7}$$

possibly resulting from application of the AWPC-CS algorithm to the corresponding shortest path problem. We want to find a set of prices $\{p'_i \mid i \in \mathcal{N}\}$ that for a given positive $\epsilon' < \epsilon$, satisfy

$$p'_i \leq a_{ij} + p'_j + \epsilon', \qquad \text{for all arcs } (i,j), \tag{3.8}$$

satisfy ϵ'-CS together with the degenerate path $P = (s)$. Thus Eq. (3.7) requires that arcs, when downhill, are downhill by at most ϵ, while Eq. (3.8) requires them to be downhill by at most ϵ'.

The idea is to start at t and sequentially proceed backwards towards s, by delineating arcs (i,j) that violate the condition (3.8) and raising the price of j and possibly the prices of some descendants of j [since increasing p_j may violate ϵ'-CS for nodes that lie downstream of j]. Thus we must check descendants of j all the way to the destination t, and raise their prices by whatever amounts are necessary to enforce the ϵ'-CS condition (3.8) on arc (i,j). Figure 3.2.6 provides an example.

The idea of successively raising the prices of the end nodes j of arcs (i,j) that violate the condition (3.8), while keeping the price of the origin s unchanged, also works for nonacyclic graphs. After a finite number of price increases, the condition (3.8) will be satisfied for all arcs. However, the number of price increases required cannot be easily predicted in the absence of special structure.

ϵ-Scaling in Graphs with Cycles

Approximate ϵ-Scaling

While the AWPC-CS algorithm maintains the ϵ-CS property if this property is initially satisfied, finding initial prices that satisfy ϵ-CS may not be easy [except when $a_{ij} \geq 0$ for all arcs (i,j), in which case we can take $p_i = 0$ for all nodes i; algorithms for the more general case are given in [Ber91],

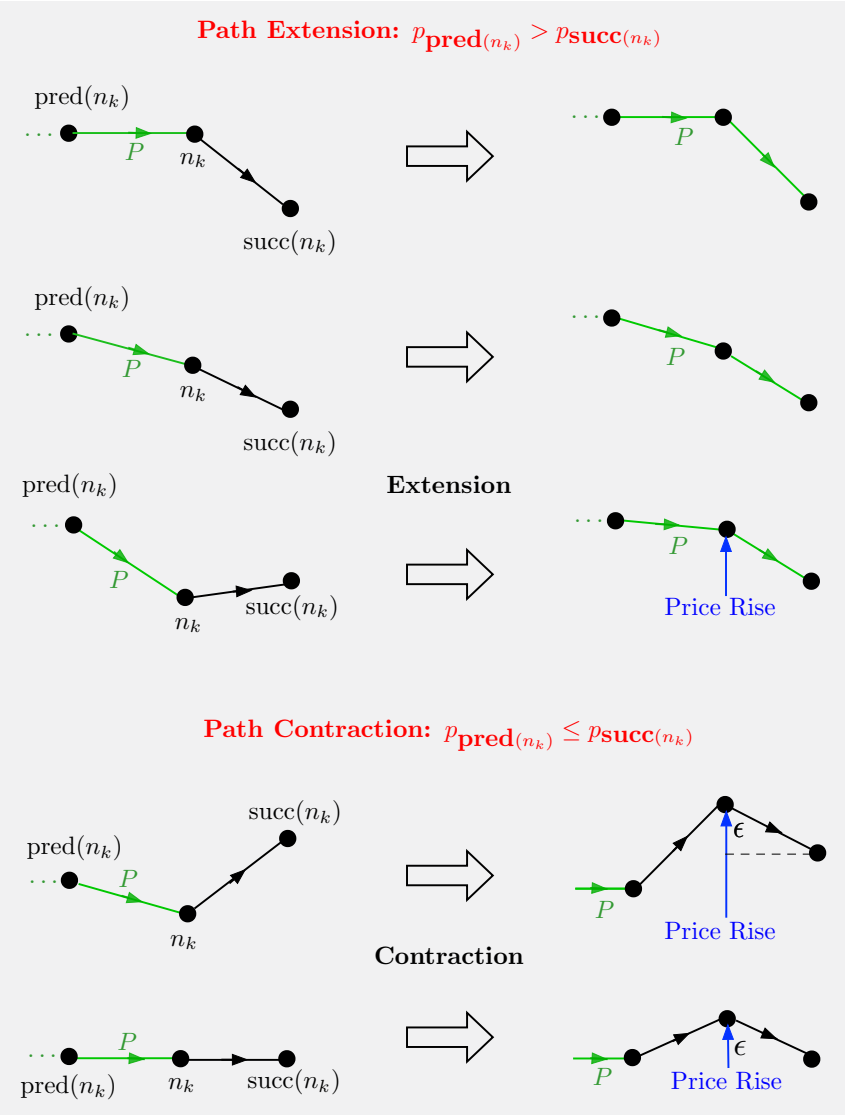


Figure 3.2.5 Illustration of an acyclic graph involving paths that start at s and end at t and arc lengths a_{ij}. A case of special interest in reinforcement learning is a tree-like structure, illustrated in the bottom figure, where nodes are grouped in layers, with arcs starting from one layer and ending at a node of the next layer, and there is a single incoming arc to each node except s and t.

Props. 6 and 7]. Moreover, operating ϵ-scaling is complicated when arc lengths change over time and on-line replanning is necessary. This moti-

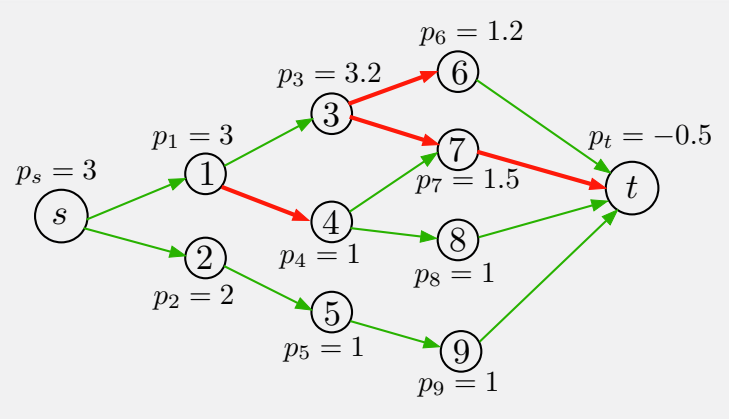


Figure 3.2.6 Illustration of ϵ-scaling in an acyclic graph, and the modifications needed to pass from the path $P = (s)$ and prices satisfying ϵ-CS to prices satisfying ϵ'-CS, with $\epsilon' < \epsilon$. All arcs have length equal to 1, and the prices p_i are shown next to the nodes i. Let $\epsilon = 1$ and $\epsilon' = 1/2$. All arcs satisfy the 1-CS condition (3.7), but arcs $(1, 4)$, $(3, 6)$, $(3, 7)$, and $(7, t)$ (shown in red) violate the 0.5-CS condition (3.8). We obtain prices satisfying 0.5-CS, by increasing the prices of the end nodes t, 6, 7, and 1 (in that order), and possibly their descendants in four iterations:

(1) Set $p_t = -0.5 \uparrow 0$.

(2) Set $p_6 = 1.2 \uparrow 1.7$ and $p_t = 0 \uparrow 0.2$.

(3) Set $p_7 = 1.5 \uparrow 1.7$ (no need to increase p_t further).

(4) Set $p_4 = 1 \uparrow 1.5$ (no need to increase p_7 or p_8, and hence also p_t).

vates a heuristic scheme to select prices that satisfy the ϵ-CS inequality

$$p_i \leq a_{ij} + p_j + \epsilon,$$

for as many arcs (i, j) as is conveniently possible, and rely on the self-correcting mechanism of the algorithm, discussed earlier, to produce a high quality solution. A similar approach may be followed for the ϵ-scaling process: it may be performed approximately, in some heuristic and computationally inexpensive way.

A reasonable approach for enforcing the ϵ'-CS property selectively is to raise the prices of just the nodes on the final path P earlier constructed by the algorithm with a larger value $\epsilon > \epsilon'$. Here, we may start from s and go forward towards t along P, while raising the prices of the nodes of P, as necessary to enforce ϵ'-CS on the arcs of P (but not on arcs outside of P). A potential benefit of this idea in some contexts is that it provides an incentive for the algorithm to explore alternative paths to P.

3.2.4 A Variant with Optimistic Extensions

Let us now discuss a variant of the AWPC algorithm, which aims to accelerate convergence by performing an extension instead of a contraction in the special case where the current path $P = (s, n_1, \ldots, n_k)$ consists of multiple nodes [i.e., $P \neq (s)$], and we have

$$p_{\text{pred}(n_k)} = a_{\text{pred}(n_k)n_k} + a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)}. \tag{3.9}$$

According to case (c2) of the AWPC algorithm, we must then perform a contraction of P to $\text{pred}(n_k)$ and raise the price of n_k to

$$a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)} + \epsilon. \tag{3.10}$$

In the variant considered in this section, called *AWPC with optimistic extensions* (AWPC-OE for short), we consider two complementary cases:

(a) $\text{succ}(n_k) \notin P$, in which case we extend P to $\text{succ}(n_k)$, and we raise the price of n_k to

$$a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)}$$

[thus making both arcs $\big(\text{pred}(n_k), n_k\big)$ and $\big(n_k, \text{succ}(n_k)\big)$ level, while maintaining the acyclicity of P].

(b) $\text{succ}(n_k) \in P$, in which case we perform a contraction of P to $\text{pred}(n_k)$, and raise the price of $\text{succ}(n_k)$ to the level (3.10), as in the AWPC algorithm.

While in this variant, the acyclicity of P is maintained at each iteration, the downhill path property as stated earlier in this section does not hold anymore, because while each arc of P is either level or downhill, it is possible that all of them are level. Still, however, the convergence proof of Prop. 3.2.1 goes through and the algorithm is valid, the critical part being that the path P remains acyclic throughout the algorithm, so that an infinite number of node contractions must be performed if the algorithm does not terminate.

For an illustration, consider the problem of Example 3.2.1. In reference to Fig. 3.2.2, the AWPC-OE algorithm will generate the same iterations as the AWPC algorithm in the first three iterations to obtain path $P = (s, 1, 2, 3)$, price vector $(p_s, p_1, p_2, p_3, p_t) = (\epsilon, \epsilon, \epsilon, 0, 0)$, $\text{pred}(3) = 2$, $\text{succ}(3) = 1$, and $p_{\text{pred}(3)} = p_{\text{succ}(3)} = \epsilon$. Then, Eq. (3.9) holds and the AWPC-OE algorithm will consider an extension to node 1, but since node 1 belongs to P, it will forego the extension, and perform a contraction to node 2, and continue exactly as the AWPC algorithm; cf. Fig. 3.2.2.

It is possible to refine the AWPC-OE algorithm for the case where Eq. (3.9) holds and there are multiple downstream neighbors j of n_k that have minimum value of $a_{n_kj} + p_j$. Then we can select one of these neighbors that does not belong to P and perform an extension, and perform a contraction

if no such neighbor can be found. Thus if there are multiple neighbors that are candidates for $\text{succ}(n_k)$, we break the tie in favor of one that does not belong to P (if one exists) and perform an extension to that neighbor. This refinement may provide additional acceleration in special types of problems, such as unweighted path construction, where multiple neighbor candidates arise frequently.

In summary, the AWPC-OE algorithm allows an extension in some cases where the AWPC algorithm performs a contraction, and is likely to terminate faster. For this it must check to make sure that no cycle is closed through an extension in the case where Eq. (3.9) holds, since the downhill path property as stated earlier may not hold.

3.3 THEORETICAL ASPECTS

Complexity analysis of the AWPC algorithm with and without ϵ-scaling.

3.4 NOTES AND SOURCES

The AWPC algorithm of this chapter was motivated by path construction algorithms given in the author's paper [Ber95a] as part of an auction algorithm for max-flow, and was also described in the book [Ber98], Section 3.3.1. The algorithms proposed here allow unrestricted choice of the initial prices, which among others, facilitates its use in reinforcement learning and on-line replanning contexts.

The AWPC algorithm also resembles an auction/shortest path algorithm that was proposed in the author's 1991 paper [Ber91]. In particular, both algorithms employ a contraction/extension mechanism for path construction. Contrary to this earlier algorithm, however, the AWPC algorithm admits arbitrary initial prices, and uses the ϵ parameter to effect larger price changes, which in turn speeds up its convergence. It produces a shortest path for sufficiently small ϵ, and it is also well-suited for an ϵ-scaling approach, which improves computational efficiency. Another important difference is that the earlier algorithm has nonpolynomial complexity, whereas the algorithm of the present paper is polynomial thanks to the use of ϵ-scaling, as discussed in Section 3.3.†

Moreover, the earlier auction/shortest path algorithm of [Ber91] requires that all cycle lengths be strictly positive rather than nonnegative,

† A polynomial variant of the 1991 auction algorithm, given in the paper [BPS95], performs very well for single origin-few destination problems with nonnegative arc lengths, but includes features that detract from the flexibility of the new algorithms of this chapter.

which is often a significant restriction. Another important difference, which affects computational efficiency, is that there is no ϵ parameter in the algorithm of [Ber91]. Indeed, this algorithm is closely related to the so called "naive auction algorithm," which is the auction algorithm for the assignment problem with $\epsilon = 0$; see [Ber91].

Actually, the APC and AWPC algorithms, can be operated with $\epsilon = 0$ provided a small change is made: in the case (c2) where $p_{\text{pred}(n_k)} = p_{\text{succ}(n_k)}$, we do an extension while setting $p_{n_k} = p_{\text{pred}(n_k)} = p_{\text{succ}(n_k)}$, rather than doing a contraction. Then the path P may not contain any uphill arcs, but it consists of just level arcs during operation of the algorithm. However, still the critical property that a cycle cannot be created through an extension is preserved, provided we assume that all cycle lengths are strictly positive (an alternative is to use the optimistic extensions mechanism, described in Section 3.2.4). Introducing a positive parameter ϵ allows for nonnegative cycle lengths, and also provides a mechanism for controlling the rate of convergence of the algorithm through the technique of ϵ-scaling, as will be discussed later in this section.