Algorithmica (2003) 36: 75–88
DOI: 10.1007/s00453-002-1008-z

Algorithmica
© 2003 Springer-Verlag New York Inc.

A Heuristic for Dijkstra's Algorithm with Many Targets and Its Use in Weighted Matching Algorithms[1]

Holger Bast,[2] Kurt Mehlhorn,[2] Guido Schäfer,[2] and Hisao Tamaki[2]

Abstract. We consider the single-source many-targets shortest-path (SSMTSP) problem in directed graphs with non-negative edge weights. A source node s and a target set T is specified and the goal is to compute a shortest path from s to a node in T. Our interest in the shortest path problem with many targets stems from its use in weighted bipartite matching algorithms. A weighted bipartite matching in a graph with n nodes on each side reduces to n SSMTSP problems, where the number of targets varies between n and 1.

The SSMTSP problem can be solved by Dijkstra's algorithm. We describe a heuristic that leads to a significant improvement in running time for the weighted matching problem; in our experiments a speed-up by up to a factor of 12 was achieved. We also present a partial analysis that gives some theoretical support for our experimental findings.

Key Words. Single-source shortest-path problem, Dijkstra's algorithm, Weighted bipartite matching problem, Assignment problem.

1. Introduction and Statement of Results.

A matching in a graph is a subset of the edges, no two of which share an endpoint. The weighted bipartite matching problem asks for the computation of a maximum weight matching in an edge-weighted bipartite graph $G = (A \dot{\cup} B, E, w)$ where the cost function $w\colon E \to \mathbb{R}$ assigns a real weight to every edge. The weight of a matching M is simply the sum of the weights of the edges in the matching, i.e., $w(M) = \sum_{e \in M} w(e)$. One may either ask for a perfect matching of maximal weight (the weighted perfect matching problem or the assignment problem) or simply for a matching of maximal weight. Both versions of the problem can be solved by solving n, $n = \max(|A|, |B|)$, single-source many-targets shortest-path (SSMTSP) problems in a derived graph, see Section 4. We describe and analyze a heuristic improvement for the SSMTSP problem which leads to a significant speed-up in LEDA's weighted bipartite matching implementation, see Table 3.

In the SSMTSP problem we are given a directed graph $G = (V, E)$ whose edges carry a non-negative cost. We use $cost(e)$ to denote the cost of an edge $e \in E$. We are also given a source node s. Every node in V is designated as either free or non-free. We are interested in finding the shortest path from s to a free node.

The SSMTSP problem is easily solved by Dijkstra's algorithm. Dijkstra's algorithm (see Section 2) maintains a tentative distance for each node and a partition of the nodes

[1] This research was partially supported by the Future and Emerging Technologies programme of the EU under Contract Number IST-1999-14186 (ALCOM-FT). A preliminary version of this paper appeared at ESA '01. The third author was funded by the Deutsche Forschungsgemeinschaft (DFG), Graduiertenkolleg *Quality Guarantees for Computer Systems*, Department of Computer Science, University of the Saarland, Germany.

[2] Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany. {bast|mehlhorn|schaefer|tamaki}@mpi-sb.mpg.de.

Received March 23, 2001; revised September 3, 2002. Communicated by H. N. Gabow.
Online publication January 13, 2003.

into settled and unsettled. At the beginning all nodes are unsettled. The algorithm operates in phases. In each phase, the unsettled node with smallest tentative distance is declared settled and its outgoing edges are relaxed in order to improve tentative distances of other unsettled nodes. The unsettled nodes are kept in a priority queue. The algorithm can be stopped once the first free node becomes settled.

We describe a heuristic improvement. The improvement maintains an upper bound for the tentative distance of free nodes and performs only queue operations with values smaller than the bound. All other queue operations are suppressed. The heuristic significantly reduces the number of queue operations and the running time of the bipartite matching algorithm, see Tables 2 and 3.

This paper is structured as follows. In Section 2 we discuss Dijkstra's algorithm for many targets and describe our heuristic. In Section 3 we give an analysis of the heuristic for random graphs and report about experiments on random graphs. In Section 4 we discuss the application to weighted bipartite matching algorithms and present our experimental findings for the matching problem.

The heuristic was first used by the third author in his jump-start routine for the general weighted matching algorithm [6], [5]. When applied to bipartite graphs, the jump-start routine computes a maximum weight matching. When we compared the running time of the jump-start routine with LEDA's bipartite matching code [4, Section 7.8], we found that the jump-start routine was consistently faster. We traced the superiority to the heuristic described in this paper.

The experiments presented in this paper were performed with the *Tool Set for Computational Experiments* (see `http://explab.sourceforge.net`). The tool provides a simple way to set up, run and analyze experiments. Moreover, it facilitates the documentation of the environment in which the experiments were run and so the experiments can be easily reproduced at a later time.

2. Dijkstra's Algorithm with Many Targets. It is useful to introduce some more notation. For a node $v \in V$, let $d(v)$ be the shortest path distance from s to v, and let $d_0 = \min\{d(v)\ ; v \text{ is free}\}$. If there is no free node reachable from s, $d_0 = +\infty$. Our goal is to compute (1) a node v_0 with $d(v_0) = d_0$ (or an indication that there is no such node), (2) the subset V' of nodes with $d(v) < d_0$, more precisely, $v \in V'$ if $d(v) < d_0$ and $d(v) \geq d_0$ if $v \notin V'$ and (3) the value $d(v)$ for every node $v \in \{v_0\} \cup V'$, i.e., a partial function $\tilde{d}$ with $\tilde{d}(v) = d(v)$ for any $v \in \{v_0\} \cup V'$. (Observe that nodes v with $d(v) = d_0$ may or may not be in V'.) We refer to the problem just described as the SSMTSP problem. It is easily solved by an adapted version of Dijkstra's algorithm as shown in Figure 1.

We maintain a priority queue *PQ* for the nodes of G. The queue is empty initially. For each node $u \in V$ we compute a tentative distance $\mathit{dist}(u)$ of a shortest path from s to u. Initially, we set $\mathit{dist}(s)$ to zero and insert the item $\langle s, 0\rangle$ into the priority queue. For each $u \in V$, $u \neq s$, we set $\mathit{dist}(u)$ to $+\infty$ (no path from s to u has been encountered yet). In the main loop we delete a node u with minimal *dist*-value from the priority queue. If u is free, we are done: $v_0 = u$ and V' is the set of nodes removed in preceding iterations. Otherwise, we relax all edges out of u. Consider an edge $e = (u, v)$ and let $c = \mathit{dist}(u) + \mathit{cost}(e)$. We check whether c is smaller than the current tentative distance

```
DIJKSTRA'S ALGORITHM (ADAPTED VERSION):
$dist(s) = 0$ and $dist(u) = +\infty$ for all $u \in V, u \neq s$
$PQ.insert(s, 0)$                                    (insert $\langle s, 0\rangle$ into $PQ$)
while not $PQ.empty()$ do
  $u = PQ.del\_min()$                 (remove node $u$ from $PQ$ with minimal priority)
  if $u$ is free then STOP fi
  RELAX ALL OUTGOING EDGES OF $u$
od

RELAX ALL OUTGOING EDGES OF $u$:
for all $e = (u, v) \in E$ do
  $c = dist(u) + cost(e)$
  if $c < dist(v)$ then
    if $dist(v) = +\infty$                               ($v$ is not contained in $PQ$)
      then $PQ.insert(v, c)$                             (insert $\langle v, c\rangle$ into $PQ$)
      else $PQ.decrease\_p(v, c)$                (decrease priority of $v$ in $PQ$ to $c$)
    fi
    $dist(v) = c$
  fi
od
```

Fig. 1. Dijkstra's algorithm adapted for many targets. When the first free node is removed from the queue, the algorithm is stopped: v_0 is the node removed last and V' consists of all non-free nodes removed from the queue.

of v. If so, we distinguish two cases: (1) If e is the first edge into v that is relaxed (this is the case iff $dist(v)$ equals $+\infty$) we insert an item $\langle v, c\rangle$ into *PQ*. (2) Otherwise, we decrease the priority of v in *PQ* to c. If a queue operation is performed, we also update $dist(v)$.

Observe that the SSMTSP problem can alternatively be solved by a single-source single-target shortest-path computation from s to a target node t, where all target nodes in T are contracted to a single target node t. The adapted version essentially does the same but without performing these contractions explicitly.

We next describe a heuristic improvement of the scheme above. Let B be the smallest *dist*-value of a free node encountered by the algorithm; $B = +\infty$ initially. We claim that queue operations $PQ.op(\cdot, c)$ with $c \geq B$ may be skipped without affecting correctness. This is clear, since the algorithm stops when the first free node is removed from the queue and since the *dist*-value of this node is certainly no larger than B. Thus all *dist*-values less than $d(v_0)$ will be computed correctly. The modified algorithm may output a different node v_0 and a different set V'. However, if all distances are pairwise distinct the same node v_0 and the same set V' as in the basic algorithm are computed. The pruning heuristic can conceivably save on queue operations, since fewer insert and decrease priority operations may be performed. Figure 2 shows the algorithm with the heuristic added.

Note that the changes which are necessary to incorporate our heuristic into the adapted version of Dijkstra's algorithm are trivial and computationally negligible. Moreover, if the underlying priority queue is stable, i.e., items with the same priority are removed from the queue in the order of their insertions, then it is clear that the heuristic will never use more queue operations than the adapted Dijkstra algorithm.

```
DIJKSTRA'S ALGORITHM WITH PRUNING HEURISTIC:
$dist(s) = 0$ and $dist(u) = +\infty$ for all $u \in V, u \neq s$
$B = +\infty$                                  (initialize upper bound to $+\infty$)
$PQ.insert(s, 0)$                              (insert $\langle s, 0\rangle$ into $PQ$)
while not $PQ.empty()$ do
  $u = PQ.del\_min()$                          (remove node $u$ from $PQ$ with minimal priority)
  if $u$ is free then STOP fi
  RELAX ALL OUTGOING EDGES OF $u$
od

RELAX ALL OUTGOING EDGES OF $u$:
for all $e = (u, v) \in E$ do
  $c = dist(u) + cost(e)$
  if $c \geq B$ then continue fi               (prune edge if bound is exceeded)
  if $v$ is free then $B = \min\{c, B\}$ fi    (try to improve bound)
  if $c < dist(v)$ then
    if $dist(v) = +\infty$                     ($v$ is not contained in $PQ$)
      then $PQ.insert(v, c)$                   (insert $\langle v, c\rangle$ into $PQ$)
      else $PQ.decrease\_p(v, c)$              (decrease priority of $v$ in $PQ$ to $c$)
    fi
    $dist(v) = c$
  fi
od
```

Fig. 2. Dijkstra's algorithm for many targets with a pruning heuristic. An upper bound B for $d(v_0)$ is maintained and queue operations $PQ.op(\cdot, c)$ with $c \geq B$ are not performed.

3. Analysis.

We perform a partial analysis of the basic and the modified version of Dijkstra's algorithm for many targets. We use n for the number of nodes, m for the expected number of edges and f for the expected number of free nodes. We assume that our graphs are random graphs in the $G(n, p)$ model with $p = m/n^2$, i.e., each of the n^2 possible edges is picked independently and uniformly at random with probability p. We use c to denote $pn = m/n$. We also assume that a node is free with probability $q = f/n$ and that edge costs are random reals between 0 and 1. We could alternatively use the model in which all graphs with m edges are equally likely and in which the free nodes form a random subset of f nodes. The results would be similar. We are mainly interested in the case where $p = c/n$ for a small constant c, say $2 \leq c \leq 10$, and q a constant, i.e., the expected number of free nodes is a fixed fraction of the nodes.

Number of Deletions from the Queue. We first analyze the number of nodes removed from the queue. If our graph were infinite and all nodes were reachable from s, the expected number would be $1/q$, namely the expected number of trials until the first head occurs in a sequence of coin tosses with success probability q. However, our graph is finite (not really a serious difference if n is large) and only a subset of the nodes is reachable from s. Observe that the probability that s has no outgoing edge is $(1 - p)^n \approx e^{-c}$. This probability is non-negligible. We proceed in two steps. We first analyze the number of nodes removed from the queue given the number R of nodes reachable from s and in a second step review results about the number R of reachable nodes.

LEMMA 1. *Let R be the number of nodes reachable from s in G and let T be the number of iterations, i.e., in iteration T the first free node is removed from the queue or there is no free node reachable from s and $T = R$. Then $\mathbf{P}(T = t \mid R = r) = (1-q)^{t-1}q$, for $1 \le t < r$, and $\mathbf{P}(T = t \mid R = r) = (1-q)^{t-1}$, for $t = r$. Moreover, for the expected number of iterations we have $\mathbf{E}[T \mid R = r] = 1/q - (1-q)^r/q$.*

PROOF. Since each node is free with probability $q = f/n$ and since the property of being free is independent from the order in which nodes are removed from the queue, we have $\mathbf{P}(T = t \mid R = r) = (1-q)^{t-1}q$ and $\mathbf{P}(T \ge t \mid R = r) = (1-q)^{t-1}$, for $1 \le t < r$. If $t = r$, $\mathbf{P}(T = t \mid R = r) = (1-q)^{t-1} = \mathbf{P}(T \ge t \mid R = r)$.

The expected number of iterations is

$$\begin{aligned}\mathbf{E}[T \mid R = r] &= \sum_{t\ge 1} \mathbf{P}(T \ge t \mid R = r) = \sum_{1 \le t < r} (1-q)^{t-1} + (1-q)^{r-1} \\ &= \frac{1-(1-q)^r}{1-(1-q)} = \frac{1}{q} - \frac{(1-q)^r}{q}. \qquad \square\end{aligned}$$

The expected number of edges relaxed is $c\mathbf{E}[(T-1) \mid R = r]$ since $T-1$ non-free nodes are removed from the queue and since the expected out-degree of every node is $c = m/n$. We conclude that the number of edges relaxed is about $((1/q)-1)(m/n)$.

Now, how many nodes are reachable from s? This quantity is analyzed in Section 10.5 of [2]. Let $\alpha > 0$ be such that $\alpha = 1 - \exp(-c\alpha)$, and let R be the number of nodes reachable from s. Then R is bounded by a constant with probability about $1-\alpha$ and is approximately αn with probability about α. More precisely, for every $\varepsilon > 0$ and $\delta > 0$, there is a t_0 such that for all sufficiently large n, we have

$$1 - \alpha - \varepsilon \le \mathbf{P}(R \le t_0) \le 1 - \alpha + \varepsilon$$

and

$$\alpha - \varepsilon \le \mathbf{P}((1-\delta)\alpha n < R < (1+\delta)\alpha n) \le \alpha + \varepsilon.$$

Table 1 indicates that small values of ε and δ work even for moderate n. For $c = 2$, we have $\alpha \approx 0.7968$. We generated 10,000 graphs with $n = 1000$ nodes and 2000 edges and determined the number of nodes reachable from a given source node s. This number was either smaller than 15 or larger than 714. The latter case occurred in $7958 \approx \alpha \cdot 10{,}000$ trials. Moreover, the average number of nodes reachable from s in the latter case was $796.5 \approx \alpha \cdot 1000 = \alpha n$.

We are only interested in the case when many nodes are reachable from s. We fix δ rather arbitrarily at 0.01 and restrict attention to the set of graphs with more than $(1-\delta)\alpha n$ nodes reachable from s. In this situation, the probability that all reachable nodes are removed from the queue is

$$(1-q)^{\alpha n} \le \exp(-\alpha n q) = \exp(-\alpha f).$$

This is less than $1/n^2$, if $c \ge 2$ and $f \ge 4 \ln n$ since $c \ge 2$ implies $\alpha > \frac{1}{2}$. We thus require our parameters to satisfy $c \ge 2$ and $f \ge 4 \ln n$ and assume that more than $(1-\delta)\alpha n$ nodes are reachable from s. We use the phrase "R is large" to refer to this assumption.

Table 1. Results of experiments.[a]

	c			
	2	5	8	8
α	0.7968	0.9930	0.9997	0.9997
MS	15	2	1	1
ML	714	981	996	1995
R	796.5	993	999.7	1999.3
F	7958	9931	9997	9995

[a] For all experiments (except the one in the last column) we used random graphs with $n = 1000$ nodes and $m = cn$ edges. For the last column we chose $n = 2000$ in order to illustrate that the dependency on n is weak. The following quantities are shown; for each value of c we performed 10^4 trials:

α: the solution of the equation $\alpha = 1 - \exp(-c\alpha)$.

MS: the maximal number of nodes reachable from s when few nodes are reachable.

ML: the minimal number of nodes reachable from s when many nodes are reachable.

R: the average number of nodes reachable from s when many nodes are reachable.

F: the number of times many nodes are reachable from s.

Number of Insertions into the Queue. We next analyze the number of insertions into the queue, first for the standard scheme, i.e., the basic algorithm of Figure 1.

LEMMA 2. *Let IS be the number of insertions into the queue in the standard scheme. Then* $\mathbf{E}[IS \mid T = t \text{ and } R \text{ is large}] \geq n - (n-1)(1-p)^{t-1}$ *for* $t < (1-\delta)\alpha n$ *and*

$$\mathbf{E}[IS \mid R \text{ is large}] \geq \frac{c(1-q)}{q+(1-q)c/n} - \frac{(1-q)c/n}{q+(1-q)c/n} + 1 - o(1) \approx \frac{c}{q} - c + 1 - o(1).$$

PROOF. We need to review some more material from Section 10.5 of [2]. Consider the following sequence of random variables:[3]

$$Y_0 = 1 \quad \text{and} \quad Y_t = Y_{t-1} + \mathbf{Bin}[n - Y_{t-1}, p] \qquad \text{for} \quad 1 \leq t \leq n$$

and let R denote the least t such that $Y_t = t$. Then R is the number of reachable nodes as a simple induction shows: observe that precisely s is reachable before the first removal and that at the time the tth node is removed from the queue, each of the $n - Y_{t-1}$ remaining (i.e., non-reached) nodes is reached with probability p. Figure 3 illustrates the process.

An inductive argument (see Section 10.5 of [2]) shows $Y_t = 1 + \mathbf{Bin}[n-1, 1-(1-p)^t]$ and hence $\mathbf{E}[Y_t] = n - (n-1)(1-p)^t$ for $0 \leq t \leq n$. We cannot directly use this result as we are interested in the process without the dashed edges. Let $E_t = (Y_0 \geq 1 \wedge \cdots \wedge Y_{t-1} \geq t)$ be the event that there are at least t reachable nodes. Then $E_{(1-\delta)\alpha n}$ is tantamount to R is large. Also, $\mathbf{E}[Y_t \mid R$ is large$]$ is the expected value of Y_t for the

[3] **Bin** [k,p] denotes the Binomial distribution with k trials and success probability p.

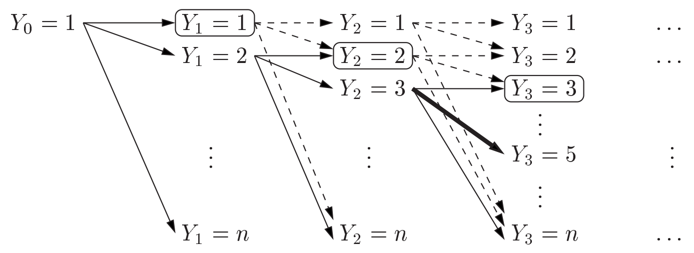

Fig. 3. The probability of the bold edge is the probability of having two successes in $n - 3$ Bernoulli trials with success probability p. We can view the process with and without the dashed edges. The process with the dashed edges corresponds to the recursive definition of the variables Y_t given in the text and the process without the dashed edges corresponds to graph exploration. In the latter case the process dies as soon as $Y_t = t$ (= a box is hit). We are interested in the latter process. The transition probabilities in the latter process differ in a non-trivial way from the transition probabilities in the original process.

process without the dashed edges. The following claim that the event R is large (= the exclusion of the dashed edges) biases Y_t towards larger values is intuitively plausible, but not at all trivial to prove.

PROPOSITION 1. *For $t \le (1-\delta)\alpha n$ we have*

$$\mathbf{E}[Y_t \mid R \textit{ is large}] \ge \mathbf{E}[Y_t] = n - (n-1)(1-p)^t.$$

PROOF. Let $N = (1-\delta)\alpha n$. Recall that $Y_0 = 1$, and $Y_t = Y_{t-1} + \mathbf{Bin}[n - Y_{t-1}, p]$ for $1 \le t \le n$. It is convenient to view the underlying probability space Ω as $\{0,1\}^{n^2}$, where entries are independently 1 with probability p. An elementary event is $\omega = (\omega_1, \ldots, \omega_n)$ where $\omega_i \in \{0,1\}^n$ and $Y_t(\omega) - Y_{t-1}(\omega)$ is the number of ones among the first $n - Y_{t-1}(\omega)$ entries of ω_t.

Let E be the event $Y_i \ge i + 1$ for $0 \le i < N$ (= the event "R is large") and let A be the event $Y_t \ge a$ for some arbitrary t and a. Both events are monotone increasing, i.e., if ω is componentwise less than or equal to ω', then $\omega \in E$ implies $\omega' \in E$ and $\omega \in A$ implies $\omega' \in A$. Thus (by Theorem 3.2 in Chapter 6 of [2])

$$\mathbf{P}(A \wedge E) \ge \mathbf{P}(A) \cdot \mathbf{P}(E) \quad \text{or} \quad \mathbf{P}(Y_t \ge a \mid R \text{ is large}) \ge \mathbf{P}(Y_t \ge a).$$

Thus

$$\mathbf{E}[Y_t \mid R \text{ is large}] \ \ge\ \mathbf{E}[Y_t]. \qquad \square$$

We can now derive our bound on the number of insertions. Let T be the number of removals from the queue. Then

$$\begin{aligned}
&\mathbf{E}[IS \mid R \text{ is large}] \\
&\quad = \mathbf{E}[IS \mid T \le (1-\delta)\alpha n \text{ and } R \text{ is large}]\mathbf{P}(T \le (1-\delta)\alpha n \mid R \text{ is large}) \\
&\qquad + \mathbf{E}[IS \mid T > (1-\delta)\alpha n \text{ and } R \text{ is large}]\mathbf{P}(T > (1-\delta)\alpha n \mid R \text{ is large}).
\end{aligned}$$

If R is large, the probability that we have more than $(1-\delta)\alpha n$ removals from the queue is $O(1/n^2)$. Thus

$$\begin{aligned}\mathbf{E}[IS \mid R \text{ is large}] &\geq \mathbf{E}[IS \mid T \leq (1-\delta)\alpha n \text{ and } R \text{ is large}](1-O(n^{-2})) \\ &\geq \mathbf{E}[IS \mid T \leq (1-\delta)\alpha n \text{ and } R \text{ is large}] - o(1).\end{aligned}$$

If R is large and $T \leq (1-\delta)\alpha n$, the procedure stops when the first free node is reached (and not because it runs out of edges). The number of insertions into the queue equals the number of nodes which are reached until the first free node is removed from the queue. Thus for $t \leq (1-\delta)\alpha n$, we obtain (recall that the outgoing edges of the free node removed are not relaxed)

$$\mathbf{E}[IS \mid T = t \text{ and } R \text{ is large}] \geq n - (n-1)(1-p)^{t-1}.$$

Thus

$$\begin{aligned}&\mathbf{E}[IS \mid R \text{ is large}] \\ &\quad \geq \sum_{t=1}^{(1-\delta)\alpha n} \mathbf{E}[IS \mid T = t \text{ and } R \text{ is large}]\mathbf{P}(T = t \mid R \text{ is large}) - o(1) \\ &\quad = \sum_{t\geq 1}(n-(n-1)(1-p)^{t-1})(1-q)^{t-1}q - o(1) \\ &\quad = n - q(n-1)\sum_{t\geq 0}(1-q)^t(1-p)^t - o(1) \\ &\quad = n - q(n-1)\frac{1}{1-(1-p)(1-q)} - o(1) \\ &\quad = n - 1 - (n-1)\frac{q}{p+q-pq} + 1 - o(1) \\ &\quad = (n-1)\frac{p-pq}{p+q-pq} + 1 - o(1) \\ &\quad = \frac{c(1-q)}{q+(1-q)c/n} - \frac{(1-q)c/n}{q+(1-q)c/n} + 1 - o(1) \\ &\quad \approx \frac{c}{q} - c + 1 - o(1).\end{aligned}$$

□

The final approximation is valid if $c/n \ll q$. The approximation makes sense intuitively: By Lemma 1, we relax the edges out of $1/q - 1$ nodes and hence relax about c times as many edges. There is hardly any sharing of targets between these edges, if n is large (and c is small). We conclude that the number of insertions into the queue is $c/q - c + 1$.

Observe that the standard scheme makes about c/q insertions into but only $1/q$ removals from the queue. This is where the refined scheme, i.e., the modified algorithm of Figure 2, saves.

Number of Nodes Inserted but Never Removed

LEMMA 3. *Let INRS be the number of nodes which are Inserted into the queue but Never Removed in the Standard scheme. Then, by the above,*

$$\mathbf{E}[\mathit{INRS} \mid R \text{ is large}] \approx \frac{c}{q} - c + 1 - \frac{1}{q} \approx \frac{c-1}{q}.$$

The standard scheme also performs some *decrease_p* operations on the nodes inserted but never removed. This number is small since the expected number of incoming edges per node is c, which we assumed to be a small constant. Observe that the expected number of insertions is basically the same as the expected number of edge relaxations.

We turn to the refined scheme. We have three kinds of savings:

- Nodes that are removed from the queue may incur fewer queue operations because they are inserted later or because some distance decreases do not lead to a queue operation. This saving is small since the number of distance decreases is small (recall that only few incoming edges per node are scanned).
- Nodes that are never removed from the queue in the standard scheme are not inserted in the refined scheme. This saving is significant and we estimate it below.
- Nodes that are never removed from the queue in the standard scheme are inserted in the refined scheme but fewer decreases of their distance labels lead to a queue operation. This saving is small for the same reason as in the first item.

We concentrate on the set of nodes that are inserted into but never removed from the queue in the standard scheme. How many of these *INRS* insertions are also performed in the refined scheme? We use *INRR* to denote their number.

LEMMA 4. *Let INRR denote the number of insertions which are also performed in the refined scheme. Then*

$$\mathbf{E}[\mathit{INRR} \mid R \text{ is large}] \le \frac{1}{q} \cdot (1 + \ln(c-1)).$$

PROOF. We first compute the expectation of *INRR* conditional on an arbitrary fixing of the edges of the graph and of the nodes removed from the queue by the standard scheme. More precisely, what is fixed in this event is the edges of the graph, the sequence of vertices removed from the queue, their distance labels and whether they are free or not.

Then what is still random in this conditional probability space? It is the weights of the edges going from a vertex removed from the queue to a vertex that is (thus inserted but) not removed from the queue, and it is whether the vertices these edges are going to are free or not. Still random are, of course, the weights of all edges with neither vertex looked at by the standard scheme, and whether these vertices are free or not.

Let $e_1 = (u_1, v_1), \ldots, e_l = (u_l, v_l)$ be the edges going from a vertex removed from the queue to a vertex that is inserted but not removed from the queue, in the order in which they are relaxed, that is, $d(u_i) \le d(u_{i+1})$, for $i = 1, \ldots, l-1$. Note that the sequence $u_1, \ldots, u_l$ may contain repetitions of the same node, corresponding to edges relaxed from the same node, whereas the $v_1, \ldots, v_l$ are all different.

The key observation is that in the conditional probability space the edge weights $w(e_1), \ldots, w(e_l)$ are still independent, and the distance label $d(u_i) + w(e_i)$ with which v_i is inserted into the queue is uniformly from $[d(u_l), d(u_i)+1]$. This is because the fixing of the nodes removed from the queue by the standard scheme implies that $d(u_i)+w(e_i) \geq d(u_l)$ but reveals nothing else about the value of $d(u_i) + w(e_i)$.

In the refined scheme e_i leads to an insertion only if $d(u_i) + w(e_i)$ is smaller than $d(u_j) + w(e_j)$ for every free v_j with $j < i$. The probability for this event is at most $1/(k+1)$, where k is the number of free v_j preceding v_i. The probability would be exactly $1/(k+1)$ if the values $d(u_h) + w(e_h)$, $1 \leq h \leq i$, were all contained in the same interval. Since the upper bound of the interval containing $d(u_h) + w(e_h)$ increases with h, the probability is at most $1/(k+1)$.

We thus obtain that, for any event E_l that fixes the edges of the graph and a sequence of l nodes removed by the standard scheme,

$$\begin{aligned}
\mathbf{E}[INRR \mid E_l] &\leq \sum_{1\leq i\leq l} \sum_{0\leq k<i} \binom{i-1}{k} q^k (1-q)^{i-1-k} \frac{1}{k+1} \\
&= \sum_{1\leq i\leq l} \frac{1}{iq} \sum_{0\leq k<i} \binom{i}{k+1} q^{k+1} (1-q)^{i-(k+1)} \\
&= \sum_{1\leq i\leq l} \frac{1}{iq} \sum_{1\leq k\leq i} \binom{i}{k} q^k (1-q)^{i-k} \\
&= \sum_{1\leq i\leq l} \frac{1}{iq} (1-(1-q)^i),
\end{aligned}$$

where the first equality follows from

$$\binom{i-1}{k} \frac{1}{k+1} = \frac{1}{i} \binom{i}{k+1}.$$

The final formula can also be interpreted intuitively. There are about iq free nodes preceding v_i and hence v_i is inserted with probability about $1/(iq)$.

In order to estimate the final sum we split the sum at a yet to be determined index i_0. For $i < i_0$, we estimate $(1-(1-q)^i) \leq iq$, and for $i \geq i_0$, we use $(1-(1-q)^i) \leq 1$. We obtain

$$\mathbf{E}[INRR \mid E_l] \leq i_0 + \frac{1}{q} \sum_{i_0\leq i\leq l} \frac{1}{i} \approx i_0 + \frac{1}{q} \ln \frac{l}{i_0}.$$

For $i_0 = 1/q$ (which minimizes the final expression[4]) we have

$$\mathbf{E}[INRR \mid E_l] \leq \frac{1}{q} \cdot (1 + \ln(lq)).$$

[4] Take the derivative with respect to $i_0 \cdots$.

Now of all the parameters that constitute E_l, this upper bound depends solely on l, the number of nodes removed by the standard scheme, so that we may conclude

$$\mathbf{E}[\mathit{INRR} \mid \mathit{INRS} = l \text{ and } R \text{ is large}] \le \frac{1}{q} \cdot (1 + \ln(lq)).$$

Since $\ln(lq)$ is a convex function of l (its first derivative is positive and its second derivative is negative), we obtain an upper bound on the expectation of *INRR* conditioned on R being large, if we replace *INRS* by its expectation. We obtain

$$\begin{aligned}\mathbf{E}[\mathit{INRR} \mid R \text{ is large}] &\le \frac{1}{q} \cdot (1 + \ln(q\mathbf{E}[\mathit{INRS} \mid R \text{ is large}])) \\ &\approx \frac{1}{q} \cdot \left(1 + \ln\left(q\frac{c-1}{q}\right)\right) = \frac{1}{q} \cdot (1 + \ln(c-1)). \qquad \square\end{aligned}$$

Number of Saved Queue Operations. We can now finally lower bound the number S of queue operations saved by the refined scheme.

THEOREM 1. *Let S denote the number of queue operations saved by the refined scheme. Then*

$$\mathbf{E}[S \mid R \textit{ is large}] \ge \frac{c}{q}\left(1 - \frac{2 + \ln(c-1)}{c}\right).$$

That is, if the refined scheme is used to solve the SSMTSP problem on random graphs drawn from the $G(n, p)$ model, with $p = c/n$ and $c = m/n$, then we are guaranteed to save at least the fraction $1 - (2 + \ln(c-1))/c$ of the queue operations performed by the standard scheme.

PROOF. By the above the saving is at least *INRS* − *INRR*. Thus

$$\mathbf{E}[S \mid R \text{ is large}] \ge \frac{c-1}{q} - \frac{1}{q}(1 + \ln(c-1)) = \frac{c}{q}\left(1 - \frac{2 + \ln(c-1)}{c}\right). \qquad \square$$

For example, if $c = 8$, we will save at least a fraction of $1 - (2 + \ln 7)/8 \approx 0.51$ of the queue operations. The actual savings are higher, see Table 2. Also, there are substantial savings even if the assumption of R being large does not hold (e.g., for $c = 2$ and $q = 0.02$).

It is interesting to observe how our randomness assumptions were used in the argument above. G is a random graph and hence the number of nodes reachable from s is either bounded or very large. The fact that a node is free with fixed probability gives us the distribution of the number of deletions from the queue. In order to estimate the savings resulting from the refined scheme we use that every node has the same chance of being free and that edge weights are random. For this part of the argument we do not need our graph to be random.

Table 2. Results of experiments.[a]

	c									
	2	2	2	5	5	5	8	8	8	8
q	0.02	0.06	0.18	0.02	0.06	0.18	0.02	0.06	0.18	0.18
D	49.60	16.40	5.51	49.33	16.72	5.50	50.22	16.79	5.61	5.53
D^*	50.00	16.67	5.56	50.00	16.67	5.56	50.00	16.67	5.56	5.56
IS	90.01	31.40	10.41	195.20	73.71	22.98	281.30	112.90	36.45	36.52
IS^*	90.16	31.35	10.02	197.60	73.57	23.25	282.30	112.30	36.13	36.77
$INRS$	40.41	15.00	4.89	145.80	56.99	17.49	231.00	96.07	30.85	30.99
$INRS^*$	40.16	14.68	4.46	147.60	56.90	17.69	232.30	95.60	30.57	31.22
$INRR$	11.00	4.00	1.00	35.00	12.00	4.00	51.00	18.00	5.00	5.00
$INRR^*$	39.05	14.56	4.34	104.10	37.13	11.99	126.80	45.78	15.03	15.15
DP_s	1.42	0.19	0.02	13.78	1.90	0.19	36.55	5.28	0.56	0.28
DP_r	0.71	0.09	0.01	2.63	0.31	0.03	4.60	0.50	0.05	0.03
Q_s	140.00	46.98	14.94	257.30	91.33	27.67	367.00	133.90	41.62	41.34
Q_r	110.40	36.12	11.52	134.50	45.33	13.97	154.40	50.85	16.00	15.77
S	29.58	10.86	3.42	122.80	46.00	13.69	212.70	83.08	25.62	25.57
S^*	1.12	0.13	0.12	43.47	19.77	5.70	105.50	49.82	15.54	16.07
P	21.12	23.11	22.87	47.74	50.37	49.50	57.94	62.03	61.55	61.85

[a]For all experiments (except the one in the last column) we used random graphs with $n = 1000$ nodes and $m = cn$ edges. For the last column we chose $n = 2000$ in order to illustrate that the dependency on n is weak. Nodes were free with probability q. The following quantities are shown; for each value of q and c we performed 10^4 trials. Trials where only a small number of nodes were reachable from s were ignored, i.e., about $(1 - \alpha) \cdot 10^4$ trials were ignored.

D: the number of deletions from the queue.

$D^* = 1/q(1 - (1 - q)^{\alpha n})$: the predicted number of deletions from the queue.

IS: the number of insertions into the queue in the standard scheme.

$IS^* = c(1 - q)/(q + (1 - q)c/n) - ((1 - q)c/n)/(q + (1 - q)c/n) + 1$: the predicted number of insertions into the queue.

$INRS$: the number of nodes inserted but never removed.

$INRS^* = IS^* - D^*$: the predicted number.

$INRR$: the number of extra nodes inserted by the refined scheme.

$INRR^* = 1/q \cdot (1 + \ln(qN^*))$: the predicted number.

DP_s: the number of decrease priority operations in the standard scheme.

DP_r: the number of decrease priority operations in the refined scheme.

Q_s: the total number of queue operations in the standard scheme.

Q_r: the total number of queue operations in the refined scheme.

$S = Q_s - Q_r$: the number of saved queue operations.

S^*: the lower bound on the number of saved queue operations.

$P = S/Q_s$: the percentage of queue operations saved.

4. Bipartite Matching Problems. Both versions of the weighted bipartite matching problem, i.e., the assignment problem and the maximum weight matching problem, can be reduced to solving n, $n = \max(|A|, |B|)$, SSMTSP problems; we discuss the reduction for the assignment problem.

A popular algorithm for the assignment problem follows the primal dual paradigm [1, Section 12.4], [4, Section 7.8], [3]. The algorithm constructs a perfect matching and a dual solution simultaneously. A dual solution is simply a function $\pi\colon V \to \mathbb{R}$ that assigns a real potential to every node. We use V to denote $A \cup B$. The algorithm maintains a matching M and a potential function π with the property that

(a) $w(e) \le \pi(a) + \pi(b)$ for every edge $e = (a, b)$,

(b) $w(e) = \pi(a) + \pi(b)$ for every edge $e = (a, b) \in M$ and
(c) $\pi(b) = 0$ for every free[5] node $b \in B$.

Initially, $M = \emptyset$, $\pi(a) = \max_{e \in E} w(e)$ for every $a \in A$ and $\pi(b) = 0$ for every $b \in B$. The algorithm stops when M is a perfect matching[6] or when it discovers that there is no perfect matching. The algorithm works in phases. In each phase the size of the matching is increased by one (or it is determined that there is no perfect matching).

A phase consists of the search for an augmenting path of minimum reduced cost. An augmenting path is a path starting at a free node in A, ending at a free node in B and using alternately edges not in M and in M. The reduced cost of an edge $e = (a, b)$ is defined as $\overline{(}e) = \pi(a) + \pi(b) - w(e)$; observe that edges in M have reduced cost zero and that all edges have non-negative reduced cost. The reduced cost of a path is simply the sum of the reduced costs of the edges contained in it. There is no need to search for augmenting paths from all free nodes in A; it suffices to search for augmenting paths from a single arbitrarily chosen free node $a_0 \in A$.

If no augmenting path starting in a_0 exists, there is no perfect matching in G and the algorithm stops. Otherwise, for every $v \in V$, let $d(v)$ be the minimal reduced cost of an alternating path from a_0 to v. Let $b_0 \in B$ be a free node in B which minimizes $d(b)$ among all free nodes b in B. We update the potential function according to the rules (we use π' to denote the new potential function):

(d) $\pi'(a) = \pi(a) - \max(d(b_0) - d(a), 0)$ for all $a \in A$,
(e) $\pi'(b) = \pi(b) + \max(d(b_0) - d(b), 0)$ for all $b \in B$.

It is easy to see that this change maintains (a)–(c) and that all edges on the least cost alternating path p from a_0 to b_0 become tight.[7] We complete the phase by switching the edges on p: matching edges on p become non-matching and non-matching edges become matching edges. This increases the size of the matching by one.[8]

A phase is tantamount to an SSMTSP problem: a_0 is the source and the free nodes are the targets. We want to determine a target (= free node) b_0 with minimal distance from a_0 and the distance values of all nodes v with $d(v) < d(b_0)$. For nodes v with $d(v) \geq d(b_0)$, there is no need to know the exact distance. It suffices to know that the distance is at least $d(b_0)$.

Table 3 shows the effect of the pruning heuristic for the bipartite matching algorithm. (The improved code will be part of LEDA Version 4.3.)

[5] A node is free if no edge in M is incident to it.
[6] It is easy to see that M has maximal weight among all perfect matchings. Observe that if M' is any perfect matching and π is any potential function such that (a) holds, then $w(M') \leq \sum_{v \in V} \pi(v)$. If (b) also holds, we have a pair (M', π) with equality and hence the matching has maximal weight (and the node potential has minimal weight among all potentials satisfying (a)).
[7] An edge is called tight if its reduced cost is zero.
[8] The correctness of the algorithm can be seen as follows. The algorithm maintains properties (a)–(c) and hence the current matching M is optimal in the following sense. Let A_m be the nodes in A that are matched. Then M is a maximal weight matching among the matchings that match the nodes in A_m and leave the nodes in $A \backslash A_m$ unmatched. Indeed, if M' is any such matching, then $w(M') \leq \sum_{a \in A_m} \pi(a) + \sum_{b \in B} \pi(b) = w(M)$, where the inequality follows from (a) and (c) and the equality follows from (b) and (c).

Table 3. Effect of the pruning heuristic.[a]

n	c	LEDA	MS	c	LEDA	MS	c	LEDA	MS
				Unit Weights					
10,000	2	0.60	0.47	5	42.51	10.80	8	93.07	8.21
20,000	2	1.32	1.03	5	152.82	39.31	8	336.24	28.20
40,000	2	2.94	2.33	5	550.54	138.88	8	1255.05	97.97
				Random Weights $[1 \cdots 1000]$					
10,000	2	0.57	0.50	5	2.33	1.41	8	11.22	4.87
20,000	2	1.20	1.05	5	5.25	3.14	8	25.41	10.79
40,000	2	2.63	2.31	5	11.09	6.80	8	56.00	23.63
				Random Weights $[1000 \cdots 1005]$					
10,000	2	0.66	0.57	5	11.42	7.02	8	20.13	11.00
20,000	2	1.39	1.22	5	36.56	22.69	8	59.36	31.59
40,000	2	3.07	2.71	5	112.05	68.29	8	181.85	99.17

[a] LEDA stands for LEDA's bipartite matching algorithm (up to version LEDA-4.2) as described in Section 7.8 of [4] and MS stands for a modified implementation with the pruning heuristic. We created random graphs with n nodes on each side and each edge is present with probability $p = c/n$. The running time is stated in CPU seconds and is an average of 10 trials.

Acknowledgements. The authors thank an anonymous referee for several helpful comments and questions. The questions revealed an error in the preliminary version of the paper.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, New York, 1992.

[3] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, **18**(1) (1986), 23–37.

[4] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, 1999.

[5] K. Mehlhorn and G. Schäfer. Implementation of $O(nm \log n)$ weighted matchings in general graphs. The power of data structures. In *Proceedings of the* 4*th International Workshop on Algorithm Engineering* (*WAE* '00). Lecture Notes in Computer Science 1982, pp. 23–38. Springer-Verlag, Berlin, 2001.

[6] G. Schäfer. Weighted Matchings in General Graphs. Master's Thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 2000.