Combinatorial Optimization at Work

Berlin 2009

Ralf Borndörfer
Zuse Institute Berlin
borndoerfer@zib.de

Contents

Chapter 1

Constraint Shortest Paths

A problem that everybody knows is the choice of a shortest travel route in a public or rail transport system, by car, or by airplane. In its basic form, this is a shortest path problem in a graph whose arcs are possible travel choices, weighted weighted by travel times. Sometimes such a shortest path is fine, but not always. In fact, it happens that a shortest path has undesirable properties such as including a lot of transfers, or being more expensive than we can afford. This leads to the consideration of one or more additonal constraints that limit the consumption of "resources" such as budget or number of transfers that we are willing to do. The most important case are linear, or, equivalently, knapsack type constraints, which add up resource consumption on a path and put a limit on it. A shortest path problem with such additional constraints is called a *(resource) constraint shortest path problem*. This lecture gives an introduction to this area by example of the constraint shortest path problem with linear resource constraints.

1.1 The Constraint Shortest Path Problem

Definition 1.1 (Chain and Path)**.** Let $D = (V, A)$ be a digraph. A *chain* in D is a sequence of arcs $v_1v_2, v_2v_3, \ldots, v_{k-1}v_k$ for some $k \in \mathbb{N}$. A chain is a *path* if the nodes $v_1, \ldots, v_k$ are pairwise different. An *st-path* is a path with $v_1 = s$ and $v_k = t$.

Definition 1.2 ((Resource) Constrained Shortest Path Problem (CSP))**.** Let the following data be given:

- a digraph $D = (V, A)$
- a *start node* $s \in V$ and an *end node* $t \in V$, $s \neq t$,
- a number of *resources* $\{1, \ldots, k\}$, $k \in \mathbb{N}$,
- *resource consumptions* $w_a^i \in \mathbb{N}_0$, $i = 1, \ldots, k$, $a \in A$, on the arcs,
- *resource limits* $d^i \in \mathbb{N}_0$, $i = 1, \ldots, k$, for each resource,
- costs $c_a \in \mathbb{N}_0$ for each arc.

The *consumption* of resource i, $i = 1, \ldots, k$ on an st-path p is $w^i(p) = \sum_{a \in p} w_a^i$. A an st-path p is *resource constrained* w.r.t. the resources $1, \ldots, k$ if it consumes not more than the limit of each resource, i.e., if $w^i(p) \leq d^i(p)$, $i = 1, \ldots, k$. The cost of an st-path p is $c(p) = \sum_{a \in p} c_a$. The *(Resource) Constraint Shortest Path Problem* (CSP) is to find a resource constrained path of minimum cost.

Notes.

- Statements on resource constraint shortest path problems are sensitive w.r.t. the groundsets from which resources and costs are drawn. Note that our definition deals with **nonnegative** and **integer** data; we will point out at the respective places when results can be generalized to a broader setting.
- Rational data can be made integer via the transformation

$$x \mapsto x \cdot N,$$

  where N is the l.c.m. of the denomiators of all rational numbers.
- Nonnegative data can be made positive via the transformation

$$x \mapsto x \cdot M + 1,$$

  where $M := |V|$ (resource limits must be scaled as $x \mapsto x \cdot M + M - 1$).
- Note that our definition does not assume that the underlying digraph is **acyclic**, i.e., does not contain directed cycles.
- We denote a CSP with k resources by $k - CSP$. For 1-CSPs, we sometimes omit the resource index, i.e., we write $w^1 = w$ and $d^1 = d$.

1.2 Complexity

We show that the CSP is NP-hard by reducing it to a well-known NP-hard problem, the knapsack problem.

Definition 1.3 (Knapsack Problem)**.** Let the following data be given:

- a set of *items* $N = \{1, \dots, n\}$,
- *weights* $w_i \in \mathbb{N}_0$, $i = 1, \dots, n$, for these items,
- a *weight limit* $d \in \mathbb{N}_0$,
- *values* $v_i \in \mathbb{N}_0$, $i = 1, \dots, n$, for the items.

The *weight* of a set of items $I \subseteq N$ is $w(I) = \sum_{i \in I} w_i$, its *value* is $v(I) = \sum_{i \in I} v_i$. The *Knapsack Problem* (KP) is to find a set of items of maximum value, with weight at most d.

Theorem 1.4 (Complexity of the CSP)**.** *The CSP is $\mathcal{NP}$-hard.*

Proof. The proof is by reduction from the knapsack problem. Let an instance of KP be given as in Definition 1.3. We construct an instance of CSP as follows:

- $V := N \cup \{0\}$
- $A := \{(i-1, i), (i-1, i)' : i = 1, \dots, n\}$
- $s := 0$, $t := n$,
- $k := 1$,
- $d^1 := d$,

- $w_a^1 := \begin{cases} w_i, & \text{if } a = (i-1,i) \\ 0, & \text{else} \end{cases}$ for all $a \in A$,
- $c_a := \begin{cases} M - v_i, & \text{if } a = (i-1,i) \\ M, & \text{else} \end{cases}$ for all $a \in A$.

It has the property that each st-paths in (V, A) contains either the arc $(i-1, i)$ or $(i-1, i)'$, $i = 1, \ldots, n$. Therefore, st-paths and item sets $I \subseteq N$ and st-paths p are in one-to-correspondence via the equivalence $i \in I \iff (i-1, i) \in p$. For such I and p holds $w^1(p) = w(I)$ and $c(p) = n \cdot M - v(I)$ and thus

$$w^1(p) \leq d^1 \iff w(I) \leq d \qquad \text{and} \qquad c(p) = \min \iff v(I) = \max.$$

□

1.3 Dynamic Programs

The CSP with positive weights can be solved by dynamic programming.

Algorithm 1 Dynamic Program (Joksch [1966])

Input: 1-CSP given as $D = (V, A)$, $s, t \in V$, $c \in \mathbb{N}_0^A$, $w \in \mathbb{N}^A$, $d \in \mathbb{N}$
Storage: Table $c_v(r)$, $v \in V$, $r = 0, \ldots, d$
Output: $c_t(d)$ = min cost of an st-path with weight at most d

for all $r = 0, \ldots, d$ **do**
  $c_s(r) \leftarrow 0$
end for
for all $v \in V$, $v \neq s$ **do**
  $c_v(0) \leftarrow \infty$
end for
for $r = 1, \ldots, d$ **do**
  for all $v \in V$, $v \neq s$ **do**
    $c_v(r) \leftarrow \min\{c_v(r-1), \min_{uv \in A, w_{uv} \leq r} c_u(r - w_{uv}) + c_{uv}\}$
  end for
end for

Theorem 1.5. *Algorithm 1 terminates correctly.*

Proof. Let $c_v(r)$ be the cost of a minimum cost sv-chain with resource consumption at most r. Then the following recursion holds:

$$\begin{aligned} &c_s(r) = 0, && r = 0, \ldots, d \\ &c_v(0) = \infty, && v \neq s \\ &c_v(r) = \min\{c_v(r-1), \min_{uv \in A, w_{uv} \leq r} c_v(r - w_{uv}) + c_{uv}\}, && v \neq s, r = 1, \ldots, d. \end{aligned}$$

This is what Algorithm 1 computes. As all costs and resource consumptions are positive, a shortest chain must be a shortest path. If no st-path exists, the recursion sets $c_t(d) = \infty$. □

Theorem 1.6. *Algorithm 1 has space complexity $O(nd)$ and time complexity $O(n + md)$, where $n = |V|$ and $m = |A|$.*

Proof. The space complexity is the size of the table $c_v(r)$, which is of size $n \cdot (d + 1)$.

For the time complexity consider the time spent in the individual lines of Algorithm 1:

$1. - 3.$	$O(d)$
$4. - 6.$	$O(n)$
$7. + 11.$	$O(d)$
$8. - 10.$	$\times O(m)$
$1. - 11.$	$O(n + md)$

□

Corollary 1.7. *The 1-CSP can be solved in pseudo-polynomial time.*

Remark 1.8.

- Algorithm 1 and Theorem 1.6 can be generalized to nonnegative data by making the cost positive with the cost scaling trick mentioned in Note 1.1. This increases time complexity to $O(nmd)$
- In acyclic digraphs, Algorithm 1 and Theorem 1.6 work with general integer costs.
- Setting $w = \mathbf{1}$ proves that the problem to find a minimum cost st-path with length (=number of arcs) **at most** d can be solved in polynomial time, namely, in $O(dm) \leq O(nm)$.
- Setting $d = n - 1$ and requiring a length of **exactly** d arcs results in a shortest Hamiltonian path problem, which is $\mathcal{NP}$-hard.

1.4 Lagrange Relaxation for the CSP in Acyclic Digraphs

Many problems in traffic applications can be modelled in terms of space-time scheduling digraphs. In such digraphs, there is a natural orientation of the arcs with respect to increasing time. Such digraphs don't have cycles.

Definition 1.9 (Acyclic Digraph)**.** A digraph $D = (V, A)$ is called *acyclic* if it does not contain a directed cycle.

Definition 1.10 (Acyclic Constrained Shortest Path Problem (ACSP))**.** A CSP with digraph D is an *acyclic constrained shortest path problem* (ACSP) if D is acyclic.

Lemma 1.11 (IP-Formulation of the ACSP)**.** *ACSP can be formulated as an integer program in the following way:*

$$\begin{array}{lrcll}
(\text{ACSP}) & \min c^{\mathrm{T}}x & & & \\
(\text{i}) & x\left(\delta^+(v)\right) - x\left(\delta^-(v)\right) & = & \delta_{st}(v) & \forall v \in V \\
(\text{ii}) & W & \leq & d & \\
(\text{iii}) & x & \geq & 0 & \\
(\text{iv}) & x & \in & \mathbb{Z}^A, &
\end{array}$$

where

$$\delta_{1n} := \begin{cases} 1, & v = s \\ -1, & v = t \\ 0, & \text{else} \end{cases}$$

and $W \in \mathbb{N}_0^{A\times k}$ is a matrix whose i-th row is w^i, i.e., $W_{i\cdot} = w^i$, $i = 1, \ldots, k$.

Proof. (ACSP) (i), (ii), and (iv) imply that x is (the incidence vector of) an integer flow of value 1. By the flow decomposition theorem, this flow can be decomposed into paths and cycles. As D is acyclic, there cannot be a cycle, i.e., x is an st-path. By (ii), it is resource constrained. □

Definition 1.12 (LP-Relaxation of the ACSP)**.** Denote by

$$\begin{array}{lrcll}
(\text{ACSP}_{LP}) & \min c^{\mathrm{T}}x & & & \\
(\text{i}) & x\left(\delta^+(v)\right) - x\left(\delta^-(v)\right) & = & \delta_{st}(v) & \forall v \in V \\
(\text{ii}) & Wx & \leq & d & \\
(\text{iii}) & x & \geq & 0 &
\end{array}$$

the *LP relaxation* of ACSP.

Definition 1.13 (Lagrange Relaxation of the ACSP)**.** Consider a feasible ACSP. For $\lambda \in \mathbb{R}_+^k$ denote by

$$\begin{array}{rcl}
\overline{c}^{\mathrm{T}} & := & \overline{c}^{\mathrm{T}}(\lambda) := c^{\mathrm{T}} + \lambda^{\mathrm{T}}W \\
f(\lambda) & := & \min_{(\text{ACSP})\ (\text{i}),\ (\text{iii}),\ (\text{iv})} \overline{c}^{\mathrm{T}}x + \lambda^{\mathrm{T}}d \\
x^*(\lambda) & := & \operatorname{argmin} f(\lambda),
\end{array}$$

where ties are broken arbitrarily. The problem

$$(\text{ACSP}_{LR}) \quad \max_{\lambda \geq 0} f(\lambda)$$

is called the *Lagrange relaxation of ACSP*; let

$$L := \max_{\lambda \geq 0} f(\lambda)$$

denote its optimal objective value.

Lemma 1.14 (Properties of ACSP_{LR})**.** *(i) $f(\lambda)$ and $x^*(\lambda)$ are well-defined.*

(ii) $L < \infty$.

(iii) $L < c^{\mathrm{T}}x$ *for every solution* x *of ACSP.*

(iv) L *is equal to the optimal value of* $ACSP_{LP}$.

Proof. (i) $f(\lambda) = \min_{P\,st\text{-path}} \overline{c}^{\mathrm{T}}\chi(P)\lambda^{\mathrm{T}}d$, i.e., $f(\lambda)$ is a minimum over a finite number of affine functions.

(ii), (iii) ACSP is feasible, i.e., there exists a solution $\overline{x}$ such that $W\overline{x}-d \leq 0$. For every such solution holds

$$\begin{array}{rcl} f(\lambda) & = & \min_{\text{(ACSP) (i), (iii), (iv)}} \overline{c}^{\mathrm{T}}x - \lambda^{\mathrm{T}}d \\ & \leq & \overline{c}^{\mathrm{T}}\overline{x} - \lambda^{\mathrm{T}}d \\ & = & c^{\mathrm{T}}\overline{x} + \lambda^{\mathrm{T}}(W\overline{x} - d) \\ & \leq & c^{\mathrm{T}}\overline{x} \\ & < & \infty. \end{array}$$

(iv) Denote by A the constraint matrix of the inequality system (ACSP) (i). Then

$$\begin{array}{rrcl} & \min c^{\mathrm{T}}x & & \\ & Ax & = & \delta_{st} \\ & -Wx & \geq & -d \\ & x & \geq & 0 \\ \\ = & \max \mu^{\mathrm{T}}\delta_{st} - \lambda^{\mathrm{T}}d & & \\ & \mu^{\mathrm{T}}A - \lambda^{\mathrm{T}}W & \leq & c^{\mathrm{T}} \\ & \lambda & \geq & 0 \\ \\ = & \max_{\lambda\geq 0} -\lambda^{\mathrm{T}}d + \max \mu^{\mathrm{T}}\delta_{st} & & \\ & \mu^{\mathrm{T}}A & \leq & c^{\mathrm{T}} + \lambda^{\mathrm{T}}W \\ \\ = & \max_{\lambda\geq 0} \min (c^{\mathrm{T}} + \lambda^{\mathrm{T}}W)x & - & \lambda^{\mathrm{T}}d \\ & Ax & = & \delta_{st} \\ & x & \geq & 0. \\ = & \max_{\lambda\geq 0} \min (c^{\mathrm{T}} + \lambda^{\mathrm{T}}W)x & - & \lambda^{\mathrm{T}}d \\ & Ax & = & \delta_{st} \\ & x & \geq & 0 \\ & x & \in & \mathbb{Z}^A. \end{array}$$

The last equality follows from total unimodularity of the constraint matrix. □

Remark 1.15. ACSP_{LR} can be solved, e.g., by subgradient algorithms.

Lemma 1.16 (Lagrangean Lower Bound for the ACSP)**.** *Consider a feasible ACSP and $\lambda \geq 0$. Denote by*

$$\begin{aligned} P_{ij} &:= \{p : \ p \text{ is } ij\text{-path in } D\} \\ c^*_{ij} &:= \min c(p), p \in P_{ij} \\ w^*_{ij} &:= \min w(p), p \in P_{ij} \\ \overline{c}^*_{ij} &:= \min \overline{c}(p), p \in P_{ij}. \end{aligned}$$

Then the following inequalities hold for every $j \in V$, $p \in P_{sj}$, and $q \in P_{jt}$ such that p, q is a resource constrained st-path:

$$\begin{aligned} &c(p) + c^*_{jn} \leq c(p) + c(q) \\ &\overline{c}(p) + \overline{c}^*_{jn} - \lambda^{\mathrm{T}} d \leq \overline{c}(p) + \overline{c}(q) - \lambda^{\mathrm{T}} d \leq c(p) + c(q) \\ &w(p) + w^*_{jn} \leq w(p) + w(q). \end{aligned}$$

Algorithm 2 Branch-and-Bound for the 1-ACSP

Input: Feasible 1-ACSP with data $D = (V, A)$, $s, t \in V$, $w \in \mathbb{N}_0^A$, $c \in \mathbb{N}^A$, $d \in \mathbb{N}$

Output: Length U of a shortest resource constrained st-path

1: **if** $w^*_s t > d$ **then**
2: output ∞ and stop
3: **end if**
4: $\lambda^* \leftarrow \operatorname{argmax}_{\lambda \geq 0} f(\lambda)$
5: $\overline{c}^{\mathrm{T}} \leftarrow c^{\mathrm{T}} - \lambda w$
6: **for all** $v \in V$ **do**
7: compute c^*_{jn}, $\overline{c}^*_{jn}$, w^*_{jn}
8: **end for**
9: $U \leftarrow c(\operatorname{argmin} w_p, p \in P_{st})$
10: $L \leftarrow \{1\}$
11: **while** $L \neq \emptyset$ **do**
12: choose $p \in L$, $L \leftarrow L \setminus \{p\}$
13: $i \leftarrow$ last node in p
14: **if** $c(p) + c^*_{in} > U$ or $w(p) + w^*_{in} > d$ or $\overline{c}(p) + \overline{c}^*_{in} - \lambda d > U$ **then**
15: next
16: **end if**
17: **if** $i = n$ **then**
18: $U \leftarrow \min U, c(p)$ and next
19: **end if**
20: **for all** $ij \in A$ **do**
21: $L \leftarrow L \cup \{pj\}$
22: **end for**
23: **end while**

1.5 Fully Polynomial Time Approximation*

The essential idea to improve Algorithm 1 in terms of running time is to work with rounded costs. There are, however, some additional technicalities. The first point is "interchanging resource consumptions and costs".

Algorithm 3 Dynamic Program (Hassin [1992])

Input: 1-CSP with data $D = (V, A)$, $s, t \in V$, $w \in \mathbb{N}_0^A$, $c \in \mathbb{N}^A$, $d \in \mathbb{N}$
Storage: Variable $\mathrm{OPT} \in \mathbb{N}_0$
Storage: Table $w_v(k)$, $v \in V$, $k = 0, \ldots, \mathrm{OPT}$ (OPT increases dynamically)
Output: OPT = min cost of an st-path with weight at most d

1: $\mathrm{OPT} \leftarrow 0$
2: $w_s(0) \leftarrow 0$
3: **for all** $v \in V$, $v \neq s$ **do**
4: $\quad w_v(0) \leftarrow \infty$
5: **end for**
6: **repeat**
7: $\quad \mathrm{OPT} \leftarrow \mathrm{OPT} + 1$
8: $\quad w_s(\mathrm{OPT}) \leftarrow 0$
9: $\quad$ **for all** $v \in V$, $v \neq s$ **do**
10: $\quad\quad w_v(\mathrm{OPT}) \leftarrow \min\{w_v(\mathrm{OPT}-1), \min_{uv \in A, c_{uv} \leq \mathrm{OPT}} w_v(\mathrm{OPT} - c_{uv}) + w_{uv}\}$
11: $\quad$ **end for**
12: **until** $w_t(\mathrm{OPT}) \leq d$ or $\mathrm{OPT} > \sum_{a \in A} c_a$
13: **if** $\mathrm{OPT} > \sum_{a \in A} c_a$ **then**
14: $\quad \mathrm{OPT} \leftarrow \infty$
15: **end if**

Theorem 1.17. *Algorithm 3 terminates correctly.*

Proof. Suppose there is an st-chain with weight at most d and $\mathrm{OPT} < \infty$ is the minimum cost of all such chains. Let further $w_v(k)$ be the weight of a minimum weight sv-chain with cost at most k, $k = 0, \ldots, \mathrm{OPT}$. Then the following recursion holds:

$$\begin{aligned}
w_s(k) &= 0 && k = 0, \ldots, \mathrm{OPT} \\
w_v(0) &= \infty && v \neq s \\
w_v(k) &= \min\{w_v(k-1), \min_{\substack{uv \in A, \\ c_{uv} \leq k}} w_v(k - c_{uv}) + w_{uv}\} && v \neq s, k = 1, \ldots, \mathrm{OPT}.
\end{aligned}$$

This is what Algorithm 3 computes. As all costs and resource consumptions are positive, a shortest chain must be a shortest path. If no st-path of weight at most d exists, the algorithm bails out with $\mathrm{OPT} = \infty$. □

Theorem 1.18. *Algorithm 3 has space complexity $O(nOPT)$ and time complexity $O(n + mOPT)$, where $n = |V|$, $m = |A|$, and OPT is the optimal cost.*

Proof. The space complexity is the size of the table $w_v(\cdot)$, which is of size $n \cdot (\mathrm{OPT} + 1)$.

For the time complexity consider the time spent in the individual lines of Algorithm 3:

$1. - 2.$	$O(1)$	
$3. - 5.$	$O(n)$	
$6. - 8. + 12.$	$O(\mathrm{OPT})$	
$9. - 11.$		$\times O(m)$
$13. - 15.$	$O(1)$	
$1. - 15.$	$O(n + m\mathrm{OPT})$	

□

We later want to apply Algorithm 3 to problems with rounded costs, which may turn out being 0. We therefore need a version that can deal with 0 costs as well.

Proposition 1.19. *Algorithm 3 can be modified to work with non-negative costs using the cost scaling trick of Note 1.1. This increases the time complexity to $O(nmOPT)$.*

We will henceforth refer to this version of Algorithm 3 as the "scaled version". It can be used to set up an approximation scheme.

Algorithm 4 Approximation Algorithm (Hassin [1992])

Input: 1-CSP with data $D = (V, A)$, $s, t \in V$, $w \in \mathbb{N}^A$, $c \in \mathbb{N}_0^A$, $d \in \mathbb{N}$ such that $1 \leq \mathrm{OPT} < \infty$
Input: Numbers $L, U \in \mathbb{N}$ such that $L \leq \mathrm{OPT} \leq U \leq 2L$
Input: Number $0 < \epsilon < 1$
Storage: Vector $\overline{c}_a$, $a \in A$, Path $\overline{P}$, Variable V
Output: $V =$ cost of an st-path with weight at most d and with the property $\mathrm{OPT} \leq V \leq (1 + \epsilon)\mathrm{OPT}$

1: **for all** $a \in A$ **do**
2: $\quad \overline{c}_a \leftarrow \lfloor \frac{c_{ij}}{L\epsilon/(n-1)} \rfloor$
3: **end for**
4: $\overline{\mathrm{OPT}} \leftarrow$ result OPT of Algorithm 3 (scaled) called for D, w, $\overline{c}$, d, s, t
5: $\overline{P} \leftarrow$ path such that $\overline{c}(P) = \overline{\mathrm{OPT}}$
6: $V \leftarrow c(\overline{P})$

Lemma 1.20. *Algorithm 4 terminates correctly.*

Proof. Let P be the minimum cost st-path with weight at most d w.r.t. the

orignal cost c. Let $\tilde{c}_a := \overline{c}_a \cdot L\epsilon/(n-1)$ for all $a \in A$. Then:

$$\begin{array}{ll} & c_a \geq \tilde{c}_a \geq c_a - L\epsilon/(n-1) \qquad \forall a \in A \\ \Longrightarrow & c(P) \geq \tilde{c}(P) \geq \tilde{c}(\overline{P}) \geq c(\overline{P}) - L\epsilon \geq c(P) - L\epsilon \\ \Longrightarrow & c(P) + L\epsilon \geq c(\overline{P}) \geq c(P) \\ \Longrightarrow & \mathrm{OPT}(1+\epsilon) \geq V \geq \mathrm{OPT}. \end{array}$$

□

Lemma 1.21. *The time complexity of Algorithm 4 is $O(\frac{mn^3}{\epsilon})$, where $n = |V|$ and $m = |A|$.*

Proof. Consider the time spent in the individual lines of Algorithm 4:

$1.-3.$	$O(m \log \frac{n}{\epsilon})$
4.	$O(n \cdot m \cdot n \max \overline{c}_a) = O(\frac{mn^3}{\epsilon})$
5.	$O(n)$
6.	$O(1)$
$1.-6.$	$O(\frac{mn^3}{\epsilon})$

For 1.–3. consider

$$\frac{c_a(n-1)}{L\epsilon} \leq \frac{U}{L}\frac{n-1}{\epsilon} \leq 2(n-1)\frac{1}{\epsilon}.$$

Hence $\overline{c}_a$, the integer part of a number $\leq 2(n-1)\epsilon$, can be computed in $O(\log \frac{n}{\epsilon})$. For 4., also note $\max \overline{c}_a \leq 2(n-1)/\epsilon$. □

In order to turn the scaled version of Algorithm 4 into an FPAS, we need to compute the bounds L and U. This can be done via some type of binary search.

Lemma 1.22. *Algorithm 5 terminates correctly.*

Proof. Case NO: If $w_t(\lceil \frac{n-1}{\epsilon} \rceil - 1) \leq d$, there is an st-path P of weight at most d such that $\overline{c}(P) \leq \lceil \frac{n-1}{\epsilon} \rceil - 1 < \frac{n-1}{\epsilon}$. Let $\tilde{c}_a := \overline{c}_a \frac{V\epsilon}{n-1}$ for all $a \in A$ such that $c_a \leq V$. Then

$$\begin{array}{ll} & c_a \geq \tilde{c}_a \geq c_a - \frac{V\epsilon}{n-1} \\ \Longrightarrow & c(P) - V\epsilon \leq \tilde{c}(P) = \overline{c}(P)\frac{V\epsilon}{n-1} < \frac{n-1}{\epsilon}\frac{V\epsilon}{n-1} = V \\ \Longrightarrow & c(P) < V(1+\epsilon). \end{array}$$

Case YES: If $w_t(\lceil \frac{n-1}{\epsilon} \rceil - 1) > d$, all st-paths P of weight at most d have cost $\overline{c}(P) > \lceil \frac{n-1}{\epsilon} \rceil - 1 \geq \frac{n-1}{\epsilon}$. This implies

$$c(P) \geq \tilde{c}(P) = \overline{c}(P)\frac{V\epsilon}{n-1} \geq \frac{n-1}{\epsilon}\frac{V\epsilon}{n-1} = V.$$

□

Algorithm 5 Approximation TEST (Hassin [1992])

Input: 1-CSP with data $D = (V, A)$, $s, t \in V$, $w \in \mathbb{N}^A$, $c \in \mathbb{N}_0^A$, $d \in \mathbb{N}$ such that $1 \leq \mathrm{OPT}$
Input: Number $V \in \mathbb{N}$ such that $1 \leq V \leq n \max c_a$
Input: Number $0 < \epsilon < 1$
Storage: Vector $\overline{c}_a$, $a \in A$
Output: Either YES: $\mathrm{OPT} \geq V$ or NO: $\mathrm{OPT} < V(1+\epsilon)$
1: **for all** $a \in A$ such that $c_a > V$ **do**
2: $A \leftarrow A \setminus \{a\}$
3: **end for**
4: **for all** $a \in A$ **do**
5: $\overline{c}_a \leftarrow \lfloor \frac{c_{ij}}{V\epsilon/(n-1)} \rfloor$
6: **end for**
7: Call Algorithm 3 (scaled) for D, w, $\overline{c}$, d, s, t
8: **if** $w_t(\lceil \frac{n-1}{\epsilon} \rceil - 1) \leq d$ **then**
9: **return** NO
10: **else**
11: **return** YES
12: **end if**

Lemma 1.23. *The time complexity of Algorithm 5 is* $O(\frac{mn^3}{\epsilon})$.

Proof.

$$\overline{c}_a = \lfloor \frac{c_{ij}}{V\epsilon/(n-1)} \rfloor \leq \frac{n-1}{\epsilon} \Longrightarrow \overline{\mathrm{OPT}} \leq \frac{(n-1)^2}{\epsilon}.$$

□

Algorithm 6 Binary Search (Hassin [1992])

Input: Numbers $1 \leq L \leq U$
Input: Unknown number $Y \in [L, U]$ given by a comparison oracle
Output: $L \leq Y \leq U$ such that $U/L \leq 2$
1: **while** $U/L > 2$ **do**
2: $X \leftarrow \sqrt{UL}$
3: **if** $Y \leq X$ **then**
4: $U \leftarrow X$
5: **else**
6: $L \leftarrow X$
7: **end if**
8: **end while**

Lemma 1.24. *Algorithm 6 terminates correctly and its time complexity is* $O(\log\log \frac{U}{L})$.

Proof.

$$
\begin{aligned}
& X/L = U/X \iff X^2 = UL \iff X = \sqrt{UL} \\
& U/X = U/\sqrt{UL} = \sqrt{U/L} \\
\iff\ & (U/L)^{(1/2)^k} < 2 \iff (1/2)^k \log U/L < \log 2 \\
\iff\ & \log(1/2)^k + \log\log U/L < \log\log 2 \\
\iff\ & k \log 1/2 < \log\log 2 - \log\log U/L \\
\iff\ & k > (\log\log 2 - \log\log U/L)/\log 1/2 \\
\implies\ & k = O(\log\log U/L).
\end{aligned}
$$

□

Algorithm 7 FPAS for 1-CSP (Hassin [1992])

Input: 1-CSP with data $D = (V, A)$, $s, t \in V$, $w \in \mathbb{N}^A$, $c \in \mathbb{N}_0^A$
Input: Number $0 < \epsilon < 1$ such that $(1+\epsilon)^4 < 2$
Storage: Numbers $L, V, U \in \mathbb{N}$
Output: V = cost of an st-path of weight at most d such thata OPT $\leq V \leq (1+\epsilon)$OPT

1: $L \leftarrow 1$, $U \leftarrow (n-1)\max c_a$
2: **while** $U/L > 2$ **do**
3: $V \leftarrow \min\{2^{(2^i)} : 2^{(2^i)} > U/L\}/2^{(2^2)}$
4: **if** TEST(VL) = YES **then**
5: $L \leftarrow VL$
6: **else**
7: $U \leftarrow VL(1+\epsilon)$
8: **end if**
9: **end while**
10: $V \leftarrow$ result of Algorithm 4 (scaled) called with D, c, w, d, s, t, L, U, ϵ

Theorem 1.25 (FPAS for 1-CSP (Hassin [1992]**.** *)] Algorithm 7 terminates correctly and has time complexity* $O(\log\log \frac{U}{L}(\frac{mn^3}{\epsilon} + (\log U)^2)))$.

Proof. In lines 1.–8. we have

$$
2^{2^i} > \frac{U}{L} \implies 2^{2^{(i-1)}} > \sqrt{\frac{U}{L}} \implies 2^{2^{(i-2)}} > (\frac{U}{L})^{1/4} \implies (\frac{U}{L})^{1/4} \leq V < \sqrt{\frac{U}{L}}.
$$

It follows

- TEST(VL) = YES: $\frac{U}{VL} \leq \frac{U}{L(U/L)^{1/4}} = (\frac{U}{L})^{\frac{3}{4}}$
- TEST(VL) = NO: $\frac{VL(1+\epsilon)}{L} \leq \sqrt{\frac{U}{L}}(1+\epsilon) \leq (\frac{U}{L})^{\frac{3}{4}}$. This holds since $1+\epsilon \leq (\frac{U}{L})^{1/4} \iff (1+\epsilon)^4 \leq \frac{U}{L} < 2$.

In both cases, the ratio $\frac{U}{L}$ is reduced to $(\frac{U}{L})$ in each iteration of the loop 2.–9. such that it takes $O(\log\log\frac{U}{L})$ iterations until the ratio falls below 2.

1.	$O(1)$		
$2.-8.$	$O(\log\log\frac{U}{L})$		
3.	$\times$	$O(\log\log\frac{U}{L}\cdot\log U)$	
4.	$\times$	$O(\frac{mn^3}{\epsilon})$	
$5.+7.$	$\times$	$(\log U)^2$	
9.	$O(\frac{mn^3}{\epsilon})$		
$1.-9.$	$O(\log\log\frac{U}{L}(\frac{mn^3}{\epsilon}+(\log U)^2)))$		

□

Corollary 1.26. *Algorithm 7 is an FPAS for the 1-CSP.*