Topics in Approximability

Konstantinos Mastakas

Department of Mathematics, University of Athens, Athens, Greece

31-1-13

Approximation Algorithms

Optimization Problem

Given an optimization problem Π and instance I of Π, let $S(I)$ denote the set of feasible solutions for I, then OPTIMUM$(I) = \min_{s \in S(I)} v(s)$(or $\max_{s \in S(I)} v(s)$) for minimization (or maximization) where $v(s)$ denotes the value of the instance

$\epsilon-$ Approximation Algorithm

An algorithm A is an $\epsilon-$ approximation algorithm for problem Π iff for every instance I, $\frac{|v(A(I)) - OPTIMUM(I)|}{\max\{OPTIMUM(I), v(A(I))\}} \leq \epsilon$, holds.

Approximation Threshold

A problem's Π approximation threshold is the $\inf\{\epsilon \geq 0 :$ there exists a polynomial $\epsilon-$ approximation algorithm$\}$

Approximation Algorithms

Optimization Problem

Given an optimization problem Π and instance I of Π, let $S(I)$ denote the set of feasible solutions for I, then OPTIMUM$(I) = \min_{s \in S(I)} v(s)$(or $\max_{s \in S(I)} v(s)$) for minimization (or maximization) where $v(s)$ denotes the value of the instance

$\epsilon-$ Approximation Algorithm

An algorithm A is an $\epsilon-$ approximation algorithm for problem Π iff for every instance I, $\frac{|v(A(I)) - OPTIMUM(I)|}{\max\{OPTIMUM(I), v(A(I))\}} \leq \epsilon$, holds.

Approximation Threshold

A problem's Π approximation threshold is the $\inf\{\epsilon \geq 0 : \text{there exists a polynomial } \epsilon - \text{approximation algorithm}\}$

Approximation Algorithms

Optimization Problem

Given an optimization problem Π and instance I of Π, let $S(I)$ denote the set of feasible solutions for I, then OPTIMUM$(I) = \min_{s \in S(I)} v(s)$(or $\max_{s \in S(I)} v(s)$) for minimization (or maximization) where $v(s)$ denotes the value of the instance

$\epsilon-$ Approximation Algorithm

An algorithm A is an $\epsilon-$ approximation algorithm for problem Π iff for every instance I, $\frac{|v(A(I)) - OPTIMUM(I)|}{\max\{OPTIMUM(I), v(A(I))\}} \leq \epsilon$, holds.

Approximation Threshold

A problem's Π approximation threshold is the $\inf\{\epsilon \geq 0 :$ there exists a polynomial $\epsilon -$ approximation algorithm$\}$

Polynomial Time Approximation Scheme (PTAS)

An algorithm A is a PTAS for the optimization problem Π, if for every instance I and $\epsilon > 0$ the relative error of $A(I, \epsilon)$ from the OPTIMUM is at most ϵ and $A(I, \epsilon)$ is calculated in time polynomially depending on $|I|$.

If $A(I, \epsilon)$ is also polynomially depending on $\frac{1}{\epsilon}$, then A is called a Fully PTAS (FPTAS).

The probabilistic relaxation of FPTAS is FPRAS, where an algorithm A is called an FPRAS for the problem Π, if for every instance of a problem, the probability, the relative error to be less than ϵ is greater than or equal to $\frac{3}{4}$.

Polynomial Time Approximation Scheme (PTAS)

An algorithm A is a PTAS for the optimization problem Π, if for every instance I and $\epsilon > 0$ the relative error of $A(I, \epsilon)$ from the OPTIMUM is at most ϵ and $A(I, \epsilon)$ is calculated in time polynomially depending on $|I|$.
If $A(I, \epsilon)$ is also polynomially depending on $\frac{1}{\epsilon}$, then A is called a Fully PTAS (FPTAS).
The probabilistic relaxation of FPTAS is FPRAS, where an algorithm A is called an FPRAS for the problem Π, if for every instance of a problem, the probability, the relative error to be less than ϵ is greater than or equal to $\frac{3}{4}$.

Polynomial Time Approximation Scheme (PTAS)

An algorithm A is a PTAS for the optimization problem Π, if for every instance I and $\epsilon > 0$ the relative error of $A(I, \epsilon)$ from the OPTIMUM is at most ϵ and $A(I, \epsilon)$ is calculated in time polynomially depending on $|I|$.
If $A(I, \epsilon)$ is also polynomially depending on $\frac{1}{\epsilon}$, then A is called a Fully PTAS (FPTAS).
The probabilistic relaxation of FPTAS is FPRAS, where an algorithm A is called an FPRAS for the problem Π, if for every instance of a problem, the probability, the relative error to be less than ϵ is greater than or equal to $\frac{3}{4}$.

FPTAS for Knapsack

Consider that there are n objects, and each of them $(1 \leq i \leq n)$ has a profit (p_i) and a weight (w_i), and we want to put a subset of the objects (the most profitable one) in a knapsack that can contain objects with weight at most W.

Pseudopolynomial algorithm

Consider P to be the maximum profit of the objects, then $\sum_i p_i \leq nP$, then for $1 \leq i \leq n$ and $0 \leq p \leq nP$ let $W(i,p)$ to be the minimum weight of a set $S \subseteq \{1, 2, \ldots, i\}$ such that $\sum_{u \in S} p_u = p$, ∞ otherwise (no set with sum of profits equals to p exists).

$W(1, p_1) = w_1$ and $W(1, p) = \infty, p \neq p_1$ and $W(i+1, p) = \min\{W(i,p), W(i, p - p_{i+1}) + w_{i+1}\}$. Using dynamic programming the problem is solved in $O(n^2 P)$

FPTAS: Consider an arbitary number b and then $p'_i = \lfloor \frac{p_i}{2^b} \rfloor$(remove the last b digits), and apply the pseudopolynomial algorithm. Now the time is $O(\frac{n^2 P}{2^b})$, and for the solution found the relative error is at most $\frac{n 2^b}{P}$. So, for every $\epsilon > 0$, b is chosen to be equal to $\lceil \log \frac{\epsilon P}{n} \rceil$ and then the execution time is $O(\frac{n^3}{\epsilon})$

FPTAS for Knapsack

Consider that there are n objects, and each of them $(1 \leq i \leq n)$ has a profit (p_i) and a weight (w_i), and we want to put a subset of the objects (the most profitable one) in a knapsack that can contain objects with weight at most W.

Pseudopolynomial algorithm

Consider P to be the maximum profit of the objects, then $\sum_i p_i \leq nP$, then for $1 \leq i \leq n$ and $0 \leq p \leq nP$ let $W(i,p)$ to be the minimum weight of a set $S \subseteq \{1,2,\ldots,i\}$ such that $\sum_{u \in S} p_u = p$, ∞ otherwise (no set with sum of profits equals to p exists).

$W(1,p_1) = w_1$ and $W(1,p) = \infty, p \neq p_1$ and $W(i+1,p) = \min\{W(i,p), W(i,p-p_{i+1}) + w_{i+1}\}$. Using dynamic programming the problem is solved in $O(n^2P)$

FPTAS: Consider an arbitary number b and then $p_i' = \lfloor \frac{p_i}{2^b} \rfloor$(remove the last b digits), and apply the pseudopolynomial algorithm. Now the time is $O(\frac{n^2P}{2^b})$, and for the solution found the relative error is at most $\frac{n2^b}{P}$. So, for every $\epsilon > 0$, b is chosen to be equal to $\lceil \log \frac{\epsilon P}{n} \rceil$ and then the execution time is $O(\frac{n^3}{\epsilon})$

FPTAS for Knapsack

Consider that there are n objects, and each of them $(1 \leq i \leq n)$ has a profit (p_i) and a weight (w_i), and we want to put a subset of the objects (the most profitable one) in a knapsack that can contain objects with weight at most W.

Pseudopolynomial algorithm

Consider P to be the maximum profit of the objects, then $\sum_i p_i \leq nP$, then for $1 \leq i \leq n$ and $0 \leq p \leq nP$ let $W(i,p)$ to be the minimum weight of a set $S \subseteq \{1,2,\dots,i\}$ such that $\sum_{u \in S} p_u = p$, ∞ otherwise (no set with sum of profits equals to p exists).

$W(1,p_1) = w_1$ and $W(1,p) = \infty, p \neq p_1$ and $W(i+1,p) = \min\{W(i,p), W(i,p-p_{i+1}) + w_{i+1}\}$. Using dynamic programming the problem is solved in $O(n^2P)$

FPTAS: Consider an arbitary number b and then $p_i' = \lfloor \frac{p_i}{2^b} \rfloor$(remove the last b digits), and apply the pseudopolynomial algorithm. Now the time is $O(\frac{n^2P}{2^b})$, and for the solution found the relative error is at most $\frac{n2^b}{P}$. So, for every $\epsilon > 0$, b is chosen to be equal to $\lceil \log \frac{\epsilon P}{n} \rceil$ and then the execution time is $O(\frac{n^3}{\epsilon})$

FPTAS for Knapsack

Consider that there are n objects, and each of them $(1 \leq i \leq n)$ has a profit (p_i) and a weight (w_i), and we want to put a subset of the objects (the most profitable one) in a knapsack that can contain objects with weight at most W.

Pseudopolynomial algorithm

Consider P to be the maximum profit of the objects, then $\sum_i p_i \leq nP$, then for $1 \leq i \leq n$ and $0 \leq p \leq nP$ let $W(i,p)$ to be the minimum weight of a set $S \subseteq \{1,2,\ldots,i\}$ such that $\sum_{u \in S} p_u = p$, ∞ otherwise (no set with sum of profits equals to p exists).
$W(1,p_1) = w_1$ and $W(1,p) = \infty, p \neq p_1$ and
$W(i+1,p) = \min\{W(i,p), W(i,p-p_{i+1}) + w_{i+1}\}$. Using dynamic programming the problem is solved in $O(n^2 P)$

FPTAS: Consider an arbitary number b and then $p'_i = \lfloor \frac{p_i}{2^b} \rfloor$(remove the last b digits), and apply the pseudopolynomial algorithm. Now the time is $O(\frac{n^2 P}{2^b})$, and for the solution found the relative error is at most $\frac{n 2^b}{P}$.

So, for every $\epsilon > 0$, b is chosen to be equal to $\lceil \log \frac{\epsilon P}{n} \rceil$ and then the execution time is $O(\frac{n^3}{\epsilon})$

FPTAS for Knapsack

Consider that there are n objects, and each of them $(1 \leq i \leq n)$ has a profit (p_i) and a weight (w_i), and we want to put a subset of the objects (the most profitable one) in a knapsack that can contain objects with weight at most W.

Pseudopolynomial algorithm

Consider P to be the maximum profit of the objects, then $\sum_i p_i \leq nP$, then for $1 \leq i \leq n$ and $0 \leq p \leq nP$ let $W(i,p)$ to be the minimum weight of a set $S \subseteq \{1, 2, \ldots, i\}$ such that $\sum_{u \in S} p_u = p$, ∞ otherwise (no set with sum of profits equals to p exists).

$W(1, p_1) = w_1$ and $W(1, p) = \infty, p \neq p_1$ and

$W(i+1, p) = \min\{W(i,p), W(i, p - p_{i+1}) + w_{i+1}\}$. Using dynamic programming the problem is solved in $O(n^2 P)$

FPTAS: Consider an arbitary number b and then $p_i' = \lfloor \frac{p_i}{2^b} \rfloor$(remove the last b digits), and apply the pseudopolynomial algorithm. Now the time is $O(\frac{n^2 P}{2^b})$, and for the solution found the relative error is at most $\frac{n 2^b}{P}$.

So, for every $\epsilon > 0$, b is chosen to be equal to $\lceil \log \frac{\epsilon P}{n} \rceil$ and then the execution time is $O(\frac{n^3}{\epsilon})$

Definition of L-Reductions

Consider the optimization problems Π_1 and Π_2, then the pair of functions (f, g) is an L-reduction from Π_1 to Π_2 iff:

1. f,g computable in logarithmic space.
2. for any instance I of Π_1, $f(I)$ is an instance of Π_2.
3. if s is a solution of $f(I)$, then $g(s)$ is a solution of I.
4. There are positive constant numbers α, β such that:
   - OPTIMUM$(f(I)) \leq \alpha \cdot$ OPTIMUM(I) and
   - If $s \in S(f(I))$, then $|\text{OPTIMUM}(I) - v(g(s))| \leq \beta|\text{OPTIMUM}(f(I)) - v(s)|$

Definition of L-Reductions

Consider the optimization problems Π_1 and Π_2, then the pair of functions (f, g) is an L-reduction from Π_1 to Π_2 iff:

1. f,g computable in logarithmic space.
2. for any instance I of Π_1, $f(I)$ is an instance of Π_2.
3. if s is a solution of $f(I)$, then $g(s)$ is a solution of I.
4. There are positive constant numbers α, β such that:
   - OPTIMUM$(f(I)) \leq \alpha \cdot$ OPTIMUM(I) and
   - If $s \in S(f(I))$, then $|\text{OPTIMUM}(I) - v(g(s))| \leq \beta |\text{OPTIMUM}(f(I)) - v(s)|$

Definition of L-Reductions

Consider the optimization problems Π_1 and Π_2, then the pair of functions (f, g) is an L-reduction from Π_1 to Π_2 iff:

1. f,g computable in logarithmic space.
2. for any instance I of Π_1, $f(I)$ is an instance of Π_2.
3. if s is a solution of $f(I)$, then $g(s)$ is a solution of I.
4. There are positive constant numbers α, β such that:
   - OPTIMUM$(f(I)) \leq \alpha \cdot$ OPTIMUM(I) and
   - If $s \in S(f(I))$, then $|\text{OPTIMUM}(I) - v(g(s))| \leq \beta |\text{OPTIMUM}(f(I)) - v(s)|$

Properties of L-Reductions

Transitivity: Consider the optimization problems Π_1, Π_2 and Π_3, if there exist (f, g) and (f', g') L-Reduction from Π_1 to Π_2 and Π_2 to Π_3, respectively, then there exist an L-Reduction $(f' \cdot f, g \cdot g')$ from Π_1 to Π_3 (where $(h \cdot h')(x) = h(h'(x))$).

Proposition: Let (f, g, α, β) an L-Reduction from Π_1 to Π_2, and there exists a polynomial time ϵ−approximation algorithm for Π_2, then there exists a polynomial time approximation algorithm for Π_1 with ratio $\frac{\alpha\beta\epsilon}{1-\epsilon}$.

Proof: Consider I to be an instance of Π_1 and $s \in S(f(I))$, the solution of the approximation algorithm of Π_2, then

$$\frac{|\text{OPTIMUM}(I) - v(g(s))|}{\max\{\text{OPTIMUM}(I), v(g(s))\}} \leq \frac{\beta|\text{OPTIMUM}(f(I)) - v(s)|}{\frac{\text{OPTIMUM}(f(I))}{\alpha}}$$

$$\leq \frac{\alpha\beta|\text{OPTIMUM}(f(I)) - v(s)|}{(1-\epsilon)\max\{\text{OPTIMUM}(f(I)), v(s)\}} \leq \frac{\alpha\beta\epsilon}{1-\epsilon}$$

Theorem: Let Π_1, Π_2 be optimization problems, then if Π_1 L-Reduces to Π_2 and there exists a PTAS for Π_2 then there exists a PTAS for Π_1.

Properties of L-Reductions

Transitivity: Consider the optimization problems Π_1, Π_2 and Π_3, if there exist (f, g) and (f', g') L-Reduction from Π_1 to Π_2 and Π_2 to Π_3, respectively, then there exist an L-Reduction $(f' \cdot f, g \cdot g')$ from Π_1 to Π_3 (where $(h \cdot h')(x) = h(h'(x))$).
Proposition: Let (f, g, α, β) an L-Reduction from Π_1 to Π_2, and there exists a polynomial time $\epsilon-$approximation algorithm for Π_2, then there exists a polynomial time approximation algorithm for Π_1 with ratio $\frac{\alpha\beta\epsilon}{1-\epsilon}$.
Proof: Consider I to be an instance of Π_1 and $s \in S(f(I))$, the solution of the approximation algorithm of Π_2, then

$$\frac{|\text{OPTIMUM}(I) - v(g(s))|}{\max\{\text{OPTIMUM}(I), v(g(s))\}} \leq \frac{\beta|\text{OPTIMUM}(f(I)) - v(s)|}{\frac{\text{OPTIMUM}(f(I))}{\alpha}}$$

$$\leq \frac{\alpha\beta|\text{OPTIMUM}(f(I)) - v(s)|}{(1-\epsilon)\max\{\text{OPTIMUM}(f(I)), v(s)\}} \leq \frac{\alpha\beta\epsilon}{1-\epsilon}$$

Theorem: Let Π_1, Π_2 be optimization problems, then if Π_1 L-Reduces to Π_2 and there exists a PTAS for Π_2 then there exists a PTAS for Π_1.

Properties of L-Reductions

Transitivity: Consider the optimization problems Π_1, Π_2 and Π_3, if there exist (f, g) and (f', g') L-Reduction from Π_1 to Π_2 and Π_2 to Π_3, respectively, then there exist an L-Reduction $(f' \cdot f, g \cdot g')$ from Π_1 to Π_3 (where $(h \cdot h')(x) = h(h'(x))$).

Proposition: Let (f, g, α, β) an L-Reduction from Π_1 to Π_2, and there exists a polynomial time $\epsilon-$approximation algorithm for Π_2, then there exists a polynomial time approximation algorithm for Π_1 with ratio $\frac{\alpha\beta\epsilon}{1-\epsilon}$.

Proof: Consider I to be an instance of Π_1 and $s \in S(f(I))$, the solution of the approximation algorithm of Π_2, then

$$\frac{|\text{OPTIMUM}(I) - v(g(s))|}{\max\{\text{OPTIMUM}(I), v(g(s))\}} \leq \frac{\beta|\text{OPTIMUM}(f(I)) - v(s)|}{\frac{\text{OPTIMUM}(f(I))}{\alpha}}$$

$$\leq \frac{\alpha\beta|\text{OPTIMUM}(f(I)) - v(s)|}{(1-\epsilon)\max\{\text{OPTIMUM}(f(I)), v(s)\}} \leq \frac{\alpha\beta\epsilon}{1-\epsilon}$$

Theorem: Let Π_1, Π_2 be optimization problems, then if Π_1 L-Reduces to Π_2 and there exists a PTAS for Π_2 then there exists a PTAS for Π_1.

Properties of L-Reductions

Transitivity: Consider the optimization problems Π_1, Π_2 and Π_3, if there exist (f, g) and (f', g') L-Reduction from Π_1 to Π_2 and Π_2 to Π_3, respectively, then there exist an L-Reduction $(f' \cdot f, g \cdot g')$ from Π_1 to Π_3 (where $(h \cdot h')(x) = h(h'(x))$).
Proposition: Let (f, g, α, β) an L-Reduction from Π_1 to Π_2, and there exists a polynomial time $\epsilon-$approximation algorithm for Π_2, then there exists a polynomial time approximation algorithm for Π_1 with ratio $\frac{\alpha\beta\epsilon}{1-\epsilon}$.
Proof: Consider I to be an instance of Π_1 and $s \in S(f(I))$, the solution of the approximation algorithm of Π_2, then

$$\frac{|\text{OPTIMUM}(I) - v(g(s))|}{\max\{\text{OPTIMUM}(I), v(g(s))\}} \leq \frac{\beta|\text{OPTIMUM}(f(I)) - v(s)|}{\frac{\text{OPTIMUM}(f(I))}{\alpha}}$$
$$\leq \frac{\alpha\beta|\text{OPTIMUM}(f(I)) - v(s)|}{(1-\epsilon)\max\{\text{OPTIMUM}(f(I)), v(s)\}} \leq \frac{\alpha\beta\epsilon}{1-\epsilon}$$

Theorem: Let Π_1, Π_2 be optimization problems, then if Π_1 L-Reduces to Π_2 and there exists a PTAS for Π_2 then there exists a PTAS for Π_1.

MAXSNP

Strict NP (SNP): is the class of the decision problems that can be expressed as: $\exists S \forall u_1 \forall u_2 \ldots \forall u_k \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k)$

for optimization problems, a more appropriate class is considered

MAXSNP$_0$: is the class of the optimization problems that can be expressed as:

$\max_S |\left\{(u_1, u_2, \ldots, u_k) \in V^k : \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k)\right\}|$

MAXSNP: An optimization problem Π belongs to the MAXSNP class iff there exists an L-Reduction from Π to an optimization problem $\Pi' \in$ MAXSNP$_0$

MAX-CUT is a MAXSNP$_0$(also, MAXSNP) problem:

$\max_{S \subseteq V} |\{(u, v) : (G(u, v) \vee G(v, u)) \wedge S(u) \wedge \neg S(v)\}|$

Theorem: Every problem belonging to MAXSNP$_0$ written as $\max_S |\{(u_1, u_2, \ldots, u_k) : \phi\}|$ has a $1 - 2^{-n_\phi}$-approximation algorithm, with n_ϕ indicating how many atomic expressions in ϕ are related to S.

MAXSNP

Strict NP (SNP): is the class of the decision problems that can be expressed as: $\exists S \forall u_1 \forall u_2 \ldots \forall u_k \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k)$
for optimization problems, a more appropriate class is considered
MAXSNP$_0$: is the class of the optimization problems that can be expressed as:
$\max_S \left| \left\{ (u_1, u_2, \ldots, u_k) \in V^k : \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k) \right\} \right|$
MAXSNP: An optimization problem Π belongs to the MAXSNP class iff there exists an L-Reduction from Π to an optimization problem $\Pi' \in$ MAXSNP$_0$
MAX-CUT is a MAXSNP$_0$(also, MAXSNP) problem:
$\max_{S \subseteq V} \left| \{ (u, v) : (G(u, v) \vee G(v, u)) \wedge S(u) \wedge \neg S(v) \} \right|$
Theorem: Every problem belonging to MAXSNP$_0$ written as $\max_S \left| \{ (u_1, u_2, \ldots, u_k) : \phi \} \right|$ has a $1 - 2^{-n_\phi}$-approximation algorithm, with n_ϕ indicating how many atomic expressions in ϕ are related to S.

MAXSNP

Strict NP (SNP): is the class of the decision problems that can be expressed as: $\exists S \forall u_1 \forall u_2 \ldots \forall u_k \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k)$
for optimization problems, a more appropriate class is considered
MAXSNP$_0$: is the class of the optimization problems that can be expressed as:
$\max_S |\left\{(u_1, u_2, \ldots, u_k) \in V^k : \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k)\right\}|$
MAXSNP: An optimization problem Π belongs to the MAXSNP class iff there exists an L-Reduction from Π to an optimization problem $\Pi' \in$ MAXSNP$_0$
MAX-CUT is a MAXSNP$_0$(also, MAXSNP) problem:
$\max_{S \subseteq V} |\{(u, v) : (G(u, v) \vee G(v, u)) \wedge S(u) \wedge \neg S(v)\}|$
Theorem: Every problem belonging to MAXSNP$_0$ written as $\max_S |\{(u_1, u_2, \ldots, u_k) : \phi\}|$ has a $1 - 2^{-n_\phi}$-approximation algorithm, with n_ϕ indicating how many atomic expressions in ϕ are related to S.

MAXSNP

Strict NP (SNP): is the class of the decision problems that can be expressed as: $\exists S \forall u_1 \forall u_2 \ldots \forall u_k \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k)$

for optimization problems, a more appropriate class is considered

MAXSNP$_0$: is the class of the optimization problems that can be expressed as:

$$\max_S |\left\{(u_1, u_2, \ldots, u_k) \in V^k : \phi(G_1, G_2, \ldots G_m, S, u_1, u_2, \ldots, u_k)\right\}|$$

MAXSNP: An optimization problem Π belongs to the MAXSNP class iff there exists an L-Reduction from Π to an optimization problem $\Pi' \in$ MAXSNP$_0$

MAX-CUT is a MAXSNP$_0$(also, MAXSNP) problem:

$$\max_{S \subseteq V} |\{(u, v) : (G(u, v) \vee G(v, u)) \wedge S(u) \wedge \neg S(v)\}|$$

Theorem: Every problem belonging to MAXSNP$_0$ written as $\max_S |\{(u_1, u_2, \ldots, u_k) : \phi\}|$ has a $1 - 2^{-n_\phi}$-approximation algorithm, with n_ϕ indicating how many atomic expressions in ϕ are related to S.

MAXSNP

Strict NP (SNP): is the class of the decision problems that can be expressed as: $\exists S \forall u_1 \forall u_2 \dots \forall u_k \phi(G_1, G_2, \dots G_m, S, u_1, u_2, \dots, u_k)$
for optimization problems, a more appropriate class is considered
MAXSNP$_0$: is the class of the optimization problems that can be expressed as:
$\max_S |\left\{(u_1, u_2, \dots, u_k) \in V^k : \phi(G_1, G_2, \dots G_m, S, u_1, u_2, \dots, u_k)\right\}|$
MAXSNP: An optimization problem Π belongs to the MAXSNP class iff there exists an L-Reduction from Π to an optimization problem $\Pi' \in$ MAXSNP$_0$
MAX-CUT is a MAXSNP$_0$(also, MAXSNP) problem:
$\max_{S \subseteq V} |\{(u, v) : (G(u, v) \vee G(v, u)) \wedge S(u) \wedge \neg S(v)\}|$
Theorem: Every problem belonging to MAXSNP$_0$ written as $\max_S |\{(u_1, u_2, \dots, u_k) : \phi\}|$ has a $1 - 2^{-n_\phi}$-approximation algorithm, with n_ϕ indicating how many atomic expressions in ϕ are related to S.

MAXSNP-Completeness

A problem Π is called MAXSNP-Complete if it belongs to MAXSNP and every other problem in MAXSNP L-Reduce to it.

If there is a PTAS for a MAXSNP-Complete problem, then for every problem in MAXSNP there is a PTAS.

MAX3SAT is MAXSNP-Complete

MAXSNP-Completeness

A problem Π is called MAXSNP-Complete if it belongs to MAXSNP and every other problem in MAXSNP L-Reduce to it.

If there is a PTAS for a MAXSNP-Complete problem, then for every problem in MAXSNP there is a PTAS.

MAX3SAT is MAXSNP-Complete

MAXSNP-Completeness

A problem Π is called MAXSNP-Complete if it belongs to MAXSNP and every other problem in MAXSNP L-Reduce to it.
If there is a PTAS for a MAXSNP-Complete problem, then for every problem in MAXSNP there is a PTAS.
MAX3SAT is MAXSNP-Complete

Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, 1995.

Vijay V. Vazirani, *Approximation Algorithm*, Springer, 2003.